



ANALISIS PERAN *FUNCTIONAL SPECIFICATION DOCUMENT* DALAM MENDUKUNG KEBERHASILAN PENGEMBANGAN PERANGKAT LUNAK: KAJIAN LITERATUR

Dimas Rhoynhan Budi Satrio¹, Umar Mukhtar², Mokhammad Afrylianto Aryo Abdi³,
Agussalim⁴

^{1,2,3,4} Sistem Informasi, Fakultas Ilmu Komputer,

Universitas Pembangunan Nasional “Veteran” Jawa Timur, Surabaya, Indonesia

E-mail: dimasbudi328@gmail.com¹, el.mochtaar@gmail.com², aryoabdi39@gmail.com³,
agussalim.si@upnjatim.ac.id⁴

ABSTRAK

Kegagalan proyek pengembangan perangkat lunak sering kali berakar pada manajemen kebutuhan yang buruk serta spesifikasi sistem yang rentan terhadap ambiguitas. Kesenjangan pemahaman antara pemangku kepentingan bisnis dan tim pengembang teknis dapat memicu pengerjaan ulang (*rework*), pembengkakan biaya, hingga kegagalan peluncuran sistem. Penelitian ini bertujuan untuk mengeksplorasi peran *Functional Specification Document* (FSD) dalam menjembatani kesenjangan tersebut melalui metode tinjauan literatur (*literature review*). Dengan mengkaji berbagai literatur dari basis data akademik, penelitian ini menganalisis fungsi utama FSD, manfaatnya dalam menekan *rework*, kontribusinya pada tahapan *Software Development Life Cycle* (SDLC), serta tantangan implementasinya di industri. Hasil sintesis literatur menunjukkan bahwa FSD berfungsi krusial sebagai sumber kebenaran tunggal (*Single Source of Truth*) yang mengikat, secara efektif mencegah pembengkakan ruang lingkup (*scope creep*), dan memberikan landasan absolut bagi fase desain arsitektur serta pengujian *Quality Assurance* (QA). Meskipun demikian, penerapan FSD di lapangan masih dihadapkan pada tantangan besar, yaitu ambiguitas akibat penggunaan bahasa alami dan kesulitan sinkronisasi pemeliharaan dokumen dalam kerangka kerja *Agile*. Penelitian ini menyimpulkan bahwa untuk mengoptimalkan peran vital FSD di era pengembangan adaptif, organisasi perlu beralih pada konsep *living documentation* dan disarankan untuk mengintegrasikan teknologi *Natural Language Processing* (NLP) guna mendeteksi kecacatan spesifikasi secara otomatis.

Kata Kunci: *Functional Specification Document*, Rekayasa Kebutuhan, Pengembangan Perangkat Lunak, Sistem Informasi

Article History

Received: Juli 2026

Reviewed: Juli 2026

Published: Juli 2026

Plagiarism Checker No
237.u873247.fh8374

Copyright : Author

Publish by : Kohehi



This work is licensed
under a [Creative
Commons Attribution-
NonCommercial 4.0
International License](https://creativecommons.org/licenses/by-nc/4.0/)

1. PENDAHULUAN

Pengembangan perangkat lunak merupakan disiplin ilmu yang sangat kompleks, di mana fase rekayasa kebutuhan (*requirements engineering*) memegang peranan esensial dalam menentukan kesuksesan proyek secara keseluruhan. Fase ini berfungsi sebagai fondasi utama untuk memastikan bahwa perangkat lunak yang dibangun secara akurat mampu menangkap perilaku, kinerja, dan batasan sistem yang diinginkan pengguna (Fariha et al., 2024). Manajemen kebutuhan yang buruk sering kali menjadi akar penyebab utama dari kegagalan

proyek teknologi informasi di industri, yang berdampak pada pembengkakan anggaran dan waktu rilis sistem (Keong & Che Embi, 2022).

Salah satu pemicu utama tingginya tingkat kegagalan tersebut adalah spesifikasi kebutuhan yang tidak terdokumentasi dengan baik. Kebutuhan yang ditulis secara tradisional menggunakan bahasa alami (*natural language*) seringkali rentan terhadap ambiguitas, yang berujung pada ketidaklengkapan dan inkonsistensi dalam rincian spesifikasi (Necula et al., 2024). Ketidaklengkapan ini memicu kesenjangan pemahaman yang serius antara pemangku kepentingan (*stakeholders*) dari sisi bisnis dan tim pengembang dari sisi teknis. Akibatnya, timbul berbagai permasalahan turunan dalam manajemen proyek, seperti miskomunikasi antar tim, hilangnya kemampuan pelacakan fitur (*traceability*), kegagalan validasi, kesalahan estimasi waktu dan biaya proyek, hingga tingginya pengerjaan ulang (*rework*) (Fariha et al., 2024), (Hussain et al., 2016).

Untuk menjembatani permasalahan komunikasi dan ambiguitas tersebut, dokumen spesifikasi yang formal dan terstruktur seperti *Functional Specification Document* (FSD) digunakan secara luas di industri (Necula et al., 2024). FSD berfungsi sebagai cetak biru krusial yang menerjemahkan ekspektasi dan proses bisnis pelanggan menjadi instruksi desain teknis yang konkret. Dokumen ini mendefinisikan dengan sangat rapi bagaimana sistem seharusnya merespons berbagai interaksi pengguna, batasan data, hingga logika bisnis yang mengikat setiap fitur. Pendekatan dokumentasi kebutuhan yang terstruktur diyakini mampu meningkatkan keselarasan pemahaman antar pihak, sehingga bermuara pada produk perangkat lunak yang jauh lebih berkualitas dan mudah dipelihara.

Namun, meskipun standar penyusunan dokumentasi telah tersedia, implementasi di lapangan tidak selalu ideal. Banyak proyek masih menghasilkan dokumentasi spesifikasi yang rentan terhadap kecacatan (*fault-prone*) akibat kurang telitnya proses elisitasi kebutuhan (Muhamad et al., 2023). Di samping itu, popularitas metodologi pengembangan adaptif (seperti *Agile*) sering kali membuat tim mengurangi porsi dokumentasi teknis formal dengan dalih kecepatan pengembangan. Ironisnya, pengabaian dokumentasi fungsional ini justru berpotensi memperbesar risiko kegagalan teknis (*technical debt*) seiring dengan bertambahnya kompleksitas fitur perangkat lunak yang dikembangkan (Keong & Che Embi, 2022).

Meskipun peran FSD diyakini sangat fundamental bagi keberlanjutan sebuah proyek, terdapat keterbatasan pada riset akademis yang mengeksplorasi secara komprehensif mengenai seberapa besar efektivitas dan kontribusi dokumen ini. Sebagian besar penelitian mutakhir lebih terfokus pada integrasi teknologi baru dalam rekayasa kebutuhan, seperti penggunaan *Natural Language Processing* (NLP) untuk otomatisasi ekstraksi (Necula et al., 2024), atau deteksi kecacatan dokumen menggunakan *machine learning* (Muhamad et al., 2023). Kajian literatur yang secara khusus memetakan dampak kualitatif dari eksistensi FSD itu sendiri dalam mendukung tahapan-tahapan rekayasa perangkat lunak masih cukup langka.

Oleh karena itu, penelitian ini menggunakan metode tinjauan literatur (*literature review*) untuk mengeksplorasi lebih dalam tentang peran *Functional Specification Document* (FSD) dalam pengembangan perangkat lunak, mencakup fungsi utamanya, manfaat penggunaannya dalam menekan *rework*, kontribusinya pada setiap tahapan pengembangan sistem, serta tantangan praktis yang dihadapi dalam penerapannya di industri.

2. TINJAUAN PUSTAKA

2.1. Rekayasa Kebutuhan (*Requirements Engineering*)

Rekayasa kebutuhan (*Requirements Engineering* atau RE) adalah fase paling awal dan fundamental dalam siklus hidup pengembangan perangkat lunak (*Software Development Life Cycle* / SDLC). Tujuan utama dari proses RE adalah untuk mengidentifikasi, menganalisis, mendokumentasikan, dan memvalidasi layanan serta batasan (*constraints*) dari sebuah sistem yang akan dikembangkan (Department of Computer Information Systems, Mutah University, Karak, Jordan & Et Al., 2021). Proses ini sangat krusial karena kesalahan atau miskonsepsi yang

terjadi pada fase definisi kebutuhan akan terakumulasi dan berlipat ganda dampaknya pada fase desain, implementasi teknis (*coding*), hingga pengujian perangkat lunak.

Kualitas produk perangkat lunak pada hasil akhir sangat bergantung pada presisi dan kelengkapan analisis kebutuhan yang didokumentasikan di awal (Wei et al., 2023). Dalam praktiknya, proses rekayasa kebutuhan berupaya untuk menjembatani kesenjangan pemahaman antara visi pengguna (klien bisnis atau *stakeholders*) dengan interpretasi teknis dari tim pengembang, sehingga ekspektasi kedua belah pihak dapat diselaraskan dan dieksekusi dengan baik (Department of Computer Information Systems, Mutah University, Karak, Jordan & Et Al., 2021).

2.2. *Functional Specification Document (FSD)*

Dalam standar rekayasa perangkat lunak dan penyusunan dokumentasi, istilah *Software Requirements Specification (SRS)* dan *Functional Specification Document (FSD)* memiliki keterikatan yang sangat erat. SRS merupakan dokumen level makro yang mencakup pemetaan keseluruhan kebutuhan sistem, baik yang bersifat fungsional maupun non-fungsional (*IEEE Recommended Practice for Software Requirements Specifications*, n.d.). Di dalam kerangka tersebut, FSD memegang peranan spesifik untuk menjabarkan kebutuhan fungsional sistem—yakni deskripsi terperinci mengenai layanan apa saja yang harus disediakan oleh perangkat lunak, bagaimana sistem harus merespons *input* tertentu, dan bagaimana perilaku antarmuka pada berbagai skenario penggunaan.

FSD berfokus pada pendefinisian "apa" (*what*) yang harus dilakukan oleh sistem untuk memenuhi proses bisnis, bukan "bagaimana" (*how*) logika arsitektur kode diimplementasikan secara teknis (Department of Computer Information Systems, Mutah University, Karak, Jordan & Et Al., 2021). Secara strategis, FSD bertindak sebagai fondasi kesepakatan formal (*baseline*) antara pelanggan dan pembuat perangkat lunak. Perumusan spesifikasi yang matang berfungsi sebagai proteksi awal terhadap kegagalan proyek, serta memberikan pijakan yang realistis untuk mengestimasi biaya operasional, jadwal penyelesaian, dan identifikasi risiko teknis di masa depan (Wei et al., 2023).

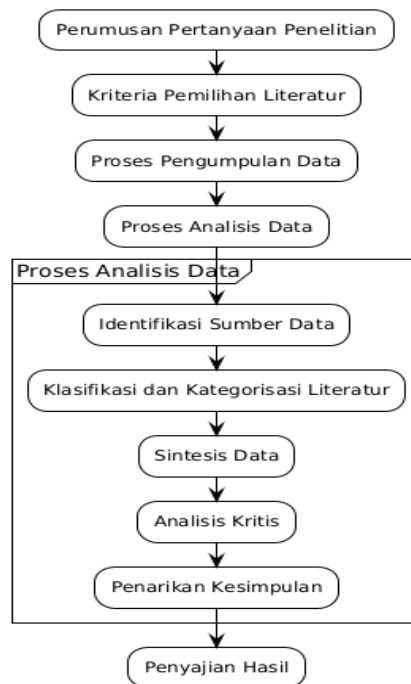
2.3. *Dinamika dan Tantangan Pendokumentasian Spesifikasi*

Meskipun penyusunan spesifikasi kebutuhan telah diakui sebagai standar esensial di industri teknologi informasi, proses ekstraksi dan pembuatannya sering kali berhadapan dengan kendala teknis dan manajerial. Penulisan spesifikasi secara manual mengonsumsi waktu yang signifikan dan membutuhkan tingkat pemahaman domain bisnis (*domain expertise*) yang sangat tinggi dari analis sistem (Wei et al., 2023).

Lebih jauh lagi, penggunaan bahasa alami (*natural language*) dalam mendeskripsikan fungsi sistem sangat rentan terhadap ambiguitas, inkonsistensi terminologi, dan ketidaklengkapan penjabaran fitur (*IEEE Recommended Practice for Software Requirements Specifications*, n.d.). Apabila kecacatan dokumentasi ini dibiarkan masuk ke dalam fase pengembangan lanjutan, hal ini akan memicu *rework* yang menguras sumber daya. Oleh karena itu, berbagai organisasi terus berupaya membakukan format dokumentasi guna meminimalkan ruang interpretasi ganda antara pihak bisnis dan tim *developer*.

3. METODE PENELITIAN

Penelitian ini menggunakan metode *literature review* untuk mengeksplorasi peran *Functional Specification Document (FSD)* dalam pengembangan perangkat lunak, dengan fokus pada fungsi utama, manfaat penggunaan, kontribusi pada tahapan pengembangan, dan tantangan yang muncul di lapangan. Metode ini dipilih karena memungkinkan peneliti untuk mengkaji berbagai sumber informasi yang telah ada untuk memahami secara utuh bagaimana FSD berdampak pada efisiensi proyek rekayasa perangkat lunak. Tahapan dalam metode penelitian ini dijelaskan sebagai berikut:



Gambar 1. Diagram alir metode penelitian

3.1. Perumusan Pertanyaan Penelitian

Penelitian dimulai dengan menyusun pertanyaan penelitian (*research question*). Pertanyaan penelitian membantu menentukan fokus penelitian sehingga kajian menjadi lebih spesifik dan terarah. Ini memastikan penelitian tidak melebar ke topik yang tidak relevan. Pertanyaan penelitian menjadi dasar dalam memilih metode penelitian yang tepat serta membantu mengarahkan proses pengumpulan dan analisis data agar relevan dengan tujuan penelitian. Pertanyaan penelitian meliputi:

Tabel 1. Research Question

Number	Research Question
RQ1	Bagaimana fungsi dan karakteristik <i>Functional Specification Document</i> (FSD) dalam siklus pengembangan perangkat lunak?
RQ2	Sejauh mana penggunaan <i>Functional Specification Document</i> memberikan manfaat dalam menekan <i>rework</i> dan mendukung keberhasilan proyek?
RQ3	Pada tahapan pengembangan sistem (SDLC) mana saja <i>Functional Specification Document</i> memberikan kontribusi paling penting?
RQ4	Apa saja tantangan manajerial dan teknis yang dihadapi oleh tim pengembang dalam menyusun dan menerapkan <i>Functional Specification Document</i> ?

3.2. Kriteria Pemilihan Literatur

Literatur yang digunakan dalam penelitian ini dipilih berdasarkan beberapa kriteria sebagai berikut:

a. Relevansi Topik

Literatur yang membahas penerapan dokumentasi kebutuhan teknis, berfokus pada pembahasan yang berkaitan dengan FSD dan *Software Requirements Specification* (SRS) guna menjawab pertanyaan penelitian.

b. Kualitas dan Kredibilitas Sumber

Literatur yang dipilih telah melalui proses *peer-review* atau berasal dari jurnal ilmiah dan prosiding konferensi bereputasi yang diakui secara akademis.

c. Jenis Penelitian

Studi yang mencakup kajian teoritis, tinjauan literatur empiris, dan studi kasus terkait implementasi dokumentasi sistem informasi.

d. Ketersediaan Data

Literatur yang menyajikan data kualitatif maupun kuantitatif yang jelas, seperti peran dokumen teknis, studi dampak dokumentasi terhadap *rework*, atau tantangan penyusunan spesifikasi.

3.3. Proses Pengumpulan Data

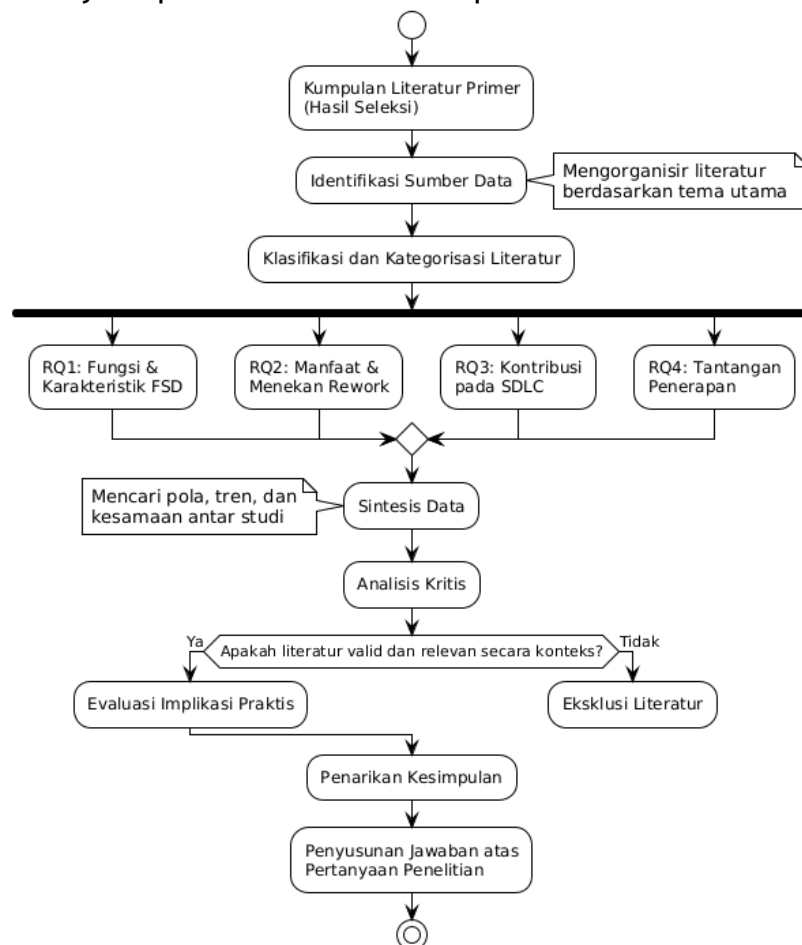
Proses pengumpulan data dilakukan melalui pencarian literatur yang relevan dengan menggunakan strategi pencarian sistematis. Pencarian dilakukan di berbagai *database* jurnal ilmiah, seperti IEEE Xplore, ACM Digital Library, Google Scholar, MDPI, ResearchGate, dan Semantic Scholar, dengan kombinasi kata kunci spesifik berikut:

- "Functional Specification Document" OR "FSD"
- "Software Requirements Specification" OR "SRS"
- "Software Engineering" AND "Documentation"
- "Requirement Engineering" AND "Challenges"

Selain itu, teknik *snowballing* digunakan untuk menemukan referensi tambahan dari daftar pustaka artikel yang relevan, dengan cara menelusuri kutipan yang ada dalam artikel yang sudah ditemukan.

3.4. Proses Analisis Data

Proses analisis data dalam penelitian ini dilakukan dengan pendekatan berikut untuk menjawab setiap pertanyaan penelitian secara komprehensif:



Gambar 2. Diagram alir proses analisis data

3.4.1. Identifikasi Sumber Data

Data yang digunakan dalam penelitian ini berasal dari berbagai sumber literatur yang telah dikumpulkan, termasuk jurnal ilmiah, artikel konferensi, dan studi kasus terkait penerapan FSD. Sumber-sumber ini diorganisir berdasarkan tema utama yang relevan dengan pertanyaan penelitian.

3.4.2. Klasifikasi dan Kategorisasi Literatur

Setelah literatur terkumpul, setiap artikel atau studi diklasifikasikan berdasarkan topik yang sesuai dengan pertanyaan penelitian. Kategorisasi utama meliputi:

- a. **Fungsi dan Karakteristik FSD (RQ1):** Fokus pada artikel yang membahas definisi operasional dan peran fundamental FSD dalam proyek perangkat lunak.
- b. **Manfaat Penggunaan FSD (RQ2):** Artikel yang mengulas bagaimana dokumentasi mencegah pembengkakan ruang lingkup (*scope creep*) dan menekan biaya pengerjaan ulang (*rework*).
- c. **Kontribusi pada Tahapan SDLC (RQ3):** Studi yang membahas integrasi FSD ke dalam fase desain arsitektur hingga fase pengujian (*Quality Assurance*).
- d. **Tantangan Penerapan FSD (RQ4):** Artikel yang membahas kendala penulisan, ambiguitas bahasa alami, serta masalah adopsi di lingkungan pengembangan adaptif (seperti *Agile*).

3.4.3. Sintesis Data

Setelah kategorisasi, langkah berikutnya adalah menyatukan temuan dari berbagai literatur yang relevan untuk setiap pertanyaan penelitian. Dalam proses ini, peneliti mencari pola, tren, dan temuan yang serupa antara berbagai studi yang ada.

3.4.4. Analisis Kritis

Setiap literatur yang digunakan dalam penelitian ini dievaluasi untuk menilai kualitas, validitas, dan relevansi temuan dalam menjawab pertanyaan penelitian. Penilaian ini mencakup konteks penerapan (apakah penelitian dilakukan dalam konteks yang relevan dengan sistem informasi) serta kesimpulan dan implikasinya secara praktis.

3.4.5. Penarikan Kesimpulan

Berdasarkan analisis yang telah dilakukan, jawaban atas setiap pertanyaan penelitian disusun secara sistematis. Jawaban ini mencakup kesimpulan dari literatur yang ditinjau serta implikasi praktis dari penerapan FSD dalam pengembangan perangkat lunak.

3.5. Penyajian Hasil

Untuk memastikan bahwa hasil penelitian ini dapat memberikan gambaran yang jelas tentang topik yang dibahas, temuan utama disajikan dengan mengelompokkan jawaban dari pertanyaan penelitian yang telah dirumuskan sebelumnya.

4. HASIL DAN PEMBAHASAN

4.1. Fungsi Utama dan Karakteristik *Functional Specification Document* (FSD) dalam Pengembangan Perangkat Lunak

4.1.1. Deskripsi *Functional Specification Document* (FSD)

Dalam industri pengembangan perangkat lunak, *Functional Specification Document* (FSD) berfungsi sebagai instrumen fundamental yang menjembatani kesenjangan komunikasi antara kebutuhan bisnis pelanggan dan implementasi teknis. FSD secara spesifik merinci "apa" (*what*) yang harus dilakukan oleh sistem, menerjemahkan harapan tingkat tinggi dari *Business Requirement Document* (BRD) menjadi alur interaksi yang sangat terukur dan terstruktur (Hussain et al., 2016).

Secara anatomi, literatur menunjukkan bahwa FSD yang komprehensif memuat beberapa elemen krusial, antara lain: definisi peran pengguna (*user roles*), deskripsi proses bisnis utama, aturan bisnis (*business rules*) yang mengikat sistem, batasan validasi *input/output* data, hingga spesifikasi integrasi dengan sistem pihak ketiga (API). Sebagai contoh operasional, dalam pengembangan platform transaksi digital, FSD mendefinisikan dengan jelas bagaimana sistem harus merespons skenario kegagalan pembayaran, merekam *log* aktivitas, dan mengatur hak

akses pengguna (Mäder & Egyed, 2015). FSD mendokumentasikan semua skenario ini murni dari perspektif fungsionalitas aplikasi, tanpa mendikte bahasa pemrograman, kerangka kerja (*framework*), atau topologi peladen (*server*) yang menjadi ranah *Technical Specification Document* (TSD).

4.1.2. Karakteristik Utama FSD

Berdasarkan sintesis dari berbagai studi rekayasa kebutuhan, dokumen FSD yang efektif untuk mendukung keberhasilan proyek harus memenuhi beberapa parameter kualitas standar internasional. Pertama, FSD harus bertindak sebagai *Single Source of Truth* (SSOT) yang bersifat lengkap (*complete*) dan konsisten, artinya tidak ada kontradiksi antar-fitur di dalam dokumen tersebut. Kedua, FSD harus memiliki tingkat ketertelusuran (*traceability*) yang tinggi, di mana setiap fungsionalitas yang diusulkan memiliki kode identifikasi unik yang dapat dilacak kembali ke asal mula kebutuhan bisnisnya. Ketiga, spesifikasi yang ditulis harus bersifat dapat diverifikasi (*verifiable*), menggunakan batasan parameter yang pasti sehingga dapat diuji secara empiris oleh tim penguji (Mäder & Egyed, 2015), (Fernández et al., 2017).

4.1.3. Pembahasan

Analisis terhadap literatur secara konsisten menunjukkan bahwa peran FSD sangat krusial dalam menciptakan landasan pemahaman (*baseline*) yang solid. Dibandingkan dengan dokumentasi tradisional yang tidak terstruktur, FSD bertindak sebagai kontrak teknis yang mengikat antara klien, analis sistem, dan tim *developer* (Hussain et al., 2016). Hal ini sejalan dengan teori rekayasa perangkat lunak klasik yang menyatakan bahwa pendefinisian spesifikasi yang matang secara drastis menurunkan beban kognitif (*cognitive load*) para pemrogram, karena mereka tidak perlu lagi menebak-nebak maksud klien saat menulis kode (Wei et al., 2023). Kesimpulannya, ketersediaan FSD yang berkualitas tinggi bukanlah sekadar syarat administratif, melainkan prasyarat operasional yang menentukan tingkat keandalan dan stabilitas arsitektur perangkat lunak yang akan dibangun.

4.2. Pengaruh Penggunaan FSD terhadap Pengurangan *Rework* dan Keberhasilan Proyek

4.2.1. Dampak terhadap Pencegahan Pembengkakan Ruang Lingkup (*Scope Creep*)

Salah satu pemicu kegagalan dan kerugian finansial terbesar dalam proyek IT adalah *scope creep*, yakni fenomena di mana pelanggan terus-menerus meminta penambahan atau modifikasi fitur di tengah fase pengerjaan karena kebutuhan awal yang tidak didefinisikan secara tajam. Penerapan FSD memitigasi risiko fatal ini dengan menetapkan perimeter ruang lingkup (*scope boundary*) sejak hari pertama. Dengan adanya FSD yang telah ditandatangani (*sign-off*), sistem menjadi "terkunci". Setiap permintaan perubahan fungsionalitas di kemudian hari harus diproses melalui prosedur *Change Request* (CR) formal yang memperhitungkan implikasi tambahan waktu dan biaya, sehingga jadwal penyelesaian proyek tetap realistis dan terukur (Fernández et al., 2017), (Inayat et al., 2015).

4.2.2. Dampak terhadap Efisiensi dan Pengurangan *Rework*

FSD berkontribusi langsung secara matematis pada efisiensi anggaran operasional dan pengurangan kerja ulang (*rework*). Literatur rekayasa perangkat lunak mengungkapkan hukum fundamental mengenai biaya perbaikan cacat (*cost of defect removal*): memperbaiki cacat konseptual pada fase penulisan dokumen spesifikasi membutuhkan biaya yang sangat minim, namun biayanya akan melonjak secara eksponensial (hingga 100 kali lipat) jika cacat tersebut baru ditemukan saat fase implementasi kode, pengujian, atau pasca-rilis (Wei et al., 2023). Dengan panduan fitur yang sangat presisi di dalam FSD, *programmer* terhindar dari asumsi pribadi yang salah sasaran, sehingga meminimalkan risiko pembongkaran ulang struktur pangkalan data (*database*) dan penulisan ulang ratusan baris kode akibat miskomunikasi (Hussain et al., 2016).

4.2.3. Pembahasan

Temuan mengenai dampak penerapan FSD ini memberikan justifikasi kuat terhadap teori efisiensi manajemen proyek. Studi empiris dari literatur terkait mengonfirmasi bahwa organisasi pengembang perangkat lunak yang menerapkan pendokumentasian spesifikasi secara

ketat mencatatkan persentase keberhasilan peluncuran (*delivery rate*) yang jauh lebih tinggi dan deviasi anggaran yang lebih rendah (Fernández et al., 2017), (Inayat et al., 2015). Hal ini membuktikan bahwa meskipun penyusunan FSD di awal proyek terasa memperlambat proses pengerjaan, investasi waktu dalam perancangan spesifikasi ini pada akhirnya terbayar lunas melalui akselerasi proses *coding* dan minimnya hambatan pada fase akhir SDLC.

4.3. Kontribusi FSD pada Tahapan *Software Development Life Cycle* (SDLC)

4.3.1. Kontribusi pada Fase Desain Arsitektur

Dalam siklus SDLC, FSD merupakan titik tolak (*starting point*) bagi berbagai fase krusial berikutnya. Pada fase desain sistem, *software architect* menggunakan FSD sebagai referensi mutlak untuk menyusun pemodelan visual, seperti *Use Case Diagram*, pemodelan *Entity-Relationship Diagram* (ERD) untuk basis data, hingga *Activity Diagram* yang memetakan alur kontrol sistem (Mäder & Egyed, 2015). Di saat yang sama, desainer *User Interface/User Experience* (UI/UX) mengandalkan batasan fungsional dari FSD untuk merancang purwarupa layar (*wireframe*) yang memastikan seluruh elemen masukan data telah terakomodasi di layar interaksi. Tanpa FSD, fase desain akan kehilangan pijakan fungsionalnya dan berisiko menghasilkan arsitektur yang tidak dapat menskalakan kebutuhan bisnis.

4.3.2. Kontribusi pada Fase Pengujian (*Quality Assurance*)

Dampak operasional paling vital dari keberadaan FSD terjadi pada fase pengujian sistem. Tim *Quality Assurance* (QA) bergantung 100% pada FSD untuk menyusun skenario uji coba (*test cases*) dan *test scripts* yang presisi. Setiap parameter, aturan batas nilai maksimal (*boundary value analysis*), dan validasi pesan *error* yang tertera pada FSD dikonversi langsung menjadi kriteria kelulusan (*acceptance criteria*) dalam pelaksanaan *User Acceptance Testing* (UAT). Jika perilaku sistem tidak menghasilkan *output* yang selaras dengan dokumen FSD, maka sistem tersebut mutlak dinyatakan cacat (*bug*). Sebaliknya, FSD melindungi *developer* dari klaim sepihak klien mengenai fitur yang dianggap "kurang" padahal sejak awal memang tidak disepakati (Keong & Che Embi, 2022).

4.3.3. Pembahasan

Mengevaluasi secara kritis keterlibatan FSD dalam siklus SDLC memperlihatkan bahwa dokumen ini bukanlah luaran pasif yang disimpan setelah disetujui, melainkan sebuah instrumen rekayasa dinamis yang mengawal proyek dari hulu ke hilir. Temuan dari tinjauan ini menegaskan bahwa validitas tahap pengujian sangat berkorelasi positif dengan seberapa rinci FSD disusun pada tahap analisis (Mäder & Egyed, 2015), (Keong & Che Embi, 2022). Pengabaian terhadap kelengkapan FSD hanya akan menghasilkan fase pengujian yang buta arah, menurunkan *test coverage*, dan pada akhirnya mereduksi keandalan aplikasi secara keseluruhan di tangan pengguna akhir.

4.4. Tantangan Manajerial dan Teknis dalam Penyusunan FSD

4.4.1. Tantangan Ambiguitas Bahasa Alami

Meskipun memiliki manfaat yang masif, literatur menyoroti bahwa penyusunan FSD di dunia nyata jarang ada yang sempurna. Masalah klasik yang mendominasi adalah munculnya "bau kebutuhan" (*requirement smells*) yang diakibatkan oleh penggunaan bahasa alami (*natural language*). Kalimat yang mengandung polisemi (kata bermakna ganda), interpretasi subjektif, dan kata sifat yang tidak dapat diukur—seperti pernyataan "sistem harus merespons dengan cepat" atau "antarmuka harus mudah digunakan"—sering kali meloloskan cacat konseptual yang fatal (Fernández et al., 2017). "Cepat" bagi klien bisa berarti 0,5 detik, namun bagi *developer* bisa berarti 3 detik. Ketidakjelasan kuantitatif semacam ini merusak prinsip *verifiable* dari FSD.

4.4.2. Kendala Pemeliharaan Dokumen dalam Metodologi *Agile*

Di era pengembangan kontemporer yang sangat didominasi oleh kerangka kerja adaptif seperti *Agile* atau *Scrum*, memelihara dokumen FSD menjadi tantangan manajerial yang sangat berat. Paradigma *Agile* menekankan prinsip "perangkat lunak yang berfungsi lebih diutamakan daripada dokumentasi yang komprehensif". Akibatnya, tim sering kali mengesampingkan

pembaruan FSD. Karena kebutuhan pasar dalam *Agile* dapat berubah secara ekstrem di setiap siklus (*sprint*), dokumen FSD yang tebal berisiko cepat usang (*obsolete*). Sinkronisasi antara versi aplikasi terbaru dengan versi FSD sering kali tertinggal, memicu munculnya utang teknis (*technical debt*) yang menyulitkan pengembang baru yang bergabung ke dalam tim (Necula et al., 2024).

4.4.3. Solusi yang Diusulkan dalam Literatur

Untuk mengatasi tantangan ambiguitas, literatur rekayasa perangkat lunak mutakhir mengusulkan adopsi teknologi *Natural Language Processing* (NLP) dan integrasi *Machine Learning*. Alat berbasis AI ini dapat memindai draf FSD untuk secara otomatis mendeteksi kata-kata yang tidak pasti (*vagueness*) atau kalimat pasif yang tidak mendefinisikan aktor sistem secara jelas (Necula et al., 2024), (Yue et al., 2011). Sementara itu, untuk mengatasi tantangan pemeliharaan dokumen dalam lingkungan *Agile*, studi merekomendasikan transisi menuju konsep *Living Documentation*. Pada pendekatan ini, spesifikasi fungsional tidak lagi ditulis dalam format dokumen Word monolitik, melainkan dipecah ke dalam unit-unit repositori digital (seperti Atlassian Confluence atau Jira) yang saling tertaut langsung dengan *repository code* dan status *testing*, memastikan visibilitas perubahan secara *real-time* (Necula et al., 2024).

4.4.4. Pembahasan

Berdasarkan analisis tantangan-tantangan tersebut, terlihat jelas bahwa praktik penyusunan FSD di masa kini membutuhkan keseimbangan strategis antara ketelitian analitis dan fleksibilitas manajerial. Dibandingkan dengan masa lalu di mana dokumentasi ditangani sepenuhnya secara manual, literatur saat ini menunjukkan pergeseran paradigma menuju otomatisasi dalam *Requirements Engineering* (Yue et al., 2011). Integrasi teknologi kecerdasan buatan untuk mengevaluasi kualitas FSD memberikan jalan keluar yang sangat inovatif bagi penyelesaian masalah ambiguitas yang berakar pada keterbatasan bahasa manusia. Kesimpulannya, agar peran vital FSD dapat terus relevan dan tidak menjadi beban administratif di industri perangkat lunak modern, proses perancangan, validasi, dan pemeliharannya harus mulai dieksekusi dengan bantuan alat otomatisasi terintegrasi.

5. KESIMPULAN DAN SARAN

Berdasarkan hasil tinjauan literatur yang telah dilakukan, *Functional Specification Document* (FSD) terbukti memegang peranan yang sangat fundamental dalam menentukan tingkat keberhasilan proyek pengembangan perangkat lunak. FSD berfungsi sebagai jembatan komunikasi yang sangat terukur antara pemangku kepentingan bisnis dan tim pengembang teknis. Dengan bertindak sebagai *Single Source of Truth* (SSOT), penerapan FSD secara disiplin mampu secara signifikan memitigasi risiko pembengkakan ruang lingkup (*scope creep*) dan menekan biaya pengerjaan ulang (*cost of rework*) yang diakibatkan oleh miskomunikasi. Dalam siklus SDLC, dokumen ini tidak hanya memberikan fondasi bagi fase desain arsitektur, tetapi juga menjadi acuan absolut bagi tim *Quality Assurance* (QA) dalam menyusun kriteria pengujian sistem.

Meskipun memberikan manfaat yang masif, penyusunan FSD masih dihadapkan pada tantangan besar, terutama terkait ambiguitas dari penggunaan bahasa alami (*natural language*) dan tingginya beban pemeliharaan dokumen di dalam lingkungan metodologi *Agile*. Oleh karena itu, disarankan bagi organisasi pengembang perangkat lunak untuk mulai bertransisi dari pendokumentasian manual menuju konsep *living documentation* yang terintegrasi. Untuk penelitian selanjutnya, sangat disarankan untuk mengkaji dan mengembangkan kerangka kerja otomatisasi berbasis *Natural Language Processing* (NLP) atau kecerdasan buatan (AI) guna mendeteksi ambiguitas dan kecacatan penulisan spesifikasi secara *real-time*, sehingga proses rekayasa kebutuhan dapat berjalan dengan jauh lebih cepat, akurat, dan adaptif terhadap perubahan pasar.

**DAFTAR PUSTAKA**

- Department of Computer Information Systems, Mutah University, Karak, Jordan, & Et Al., A.-M. (2021). Constructing a software requirements specification and design for electronic IT news magazine system. *International Journal of ADVANCED AND APPLIED SCIENCES*, 8(11), 104-118. <https://doi.org/10.21833/ijaas.2021.11.014>
- Fariha, A., Alwidian, S., & Azim, A. (2024). A Systematic Literature Review on Requirements Engineering and Maintenance for Embedded Software. *IEEE Access*, 12, 114263-114279. <https://doi.org/10.1109/ACCESS.2024.3443271>
- Fernández, D. M., Wagner, S., Kalinowski, M., Felderer, M., Mafra, P., Vetrò, A., Conte, T., Christiansson, M.-T., Greer, D., Lassenius, C., Männistö, T., Nayabi, M., Oivo, M., Penzenstadler, B., Pfahl, D., Prikladnicki, R., Ruhe, G., Schekelmann, A., Sen, S., ... Wieringa, R. (2017). Naming the pain in requirements engineering: Contemporary problems, causes, and effects in practice. *Empirical Software Engineering*, 22(5), 2298-2338. <https://doi.org/10.1007/s10664-016-9451-7>
- Hussain, A., Mkpojiogu, E., & Kamal, F. (2016). The Role of Requirements in the Success or Failure of Software Projects. *EJ Econjournals*, 6, 6-7.
- IEEE Recommended Practice for Software Requirements Specifications*. (n.d.). IEEE. <https://doi.org/10.1109/IEEESTD.1998.88286>
- Inayat, I., Salim, S. S., Marczak, S., Daneva, M., & Shamshirband, S. (2015). A systematic literature review on agile requirements engineering practices and challenges. *Computers in Human Behavior*, 51, 915-929. <https://doi.org/10.1016/j.chb.2014.10.046>
- Keong, S. L. M., & Che Embi, Z. (2022). A Systematic Review on Non-Functional Requirements Documentation in Agile Methodology. *Journal of Informatics and Web Engineering*, 1(2), 19. <https://doi.org/10.33093/jiwe.2022.1.2.2>
- Mäder, P., & Egyed, A. (2015). Do developers benefit from requirements traceability when evolving and maintaining a software system? *Empirical Softw. Engg.*, 20(2), 413-441. <https://doi.org/10.1007/s10664-014-9314-z>
- Muhamad, F. N. J., Ab Hamid, S. H., Subramaniam, H., Abdul Rashid, R., & Fahmi, F. (2023). Fault-Prone Software Requirements Specification Detection Using Ensemble Learning for Edge/Cloud Applications. *Applied Sciences*, 13(14), 8368. <https://doi.org/10.3390/app13148368>
- Necula, S.-C., Dumitriu, F., & Greavu-Şerban, V. (2024). A Systematic Literature Review on Using Natural Language Processing in Software Requirements Engineering. *Electronics*, 13(11), 2055. <https://doi.org/10.3390/electronics13112055>
- Wei, X., Wang, Z., & Yang, S. (2023). An Automatic Generation and Verification Method of Software Requirements Specification. *Electronics*, 12(12), 2734. <https://doi.org/10.3390/electronics12122734>
- Yue, T., Briand, L. C., & Labiche, Y. (2011). A systematic review of transformation approaches between user requirements and analysis models. *Requirements Engineering*, 16(2), 75-99. <https://doi.org/10.1007/s00766-010-0111-y>