REKAYASA KEBUTUHAN DAN PEMODELAN SISTEM PERANGKAT LUNAK: KONSEP, TEORI DAN PRAKTIK DENGAN UML

Ade Maulana
Irmawati
Nanang Durahman
Wirawan Istiono
Johni S Pasaribu
Alexander Waworuntu
Green Ferry Mandias
Raymond Sunardi Oetama
Arief Yanto Rukmana

GET PRESS INDONESIA

REKAYASA KEBUTUHAN DAN PEMODELAN SISTEM PERANGKAT LUNAK: KONSEP, TEORI DAN PRAKTIK DENGAN UML

Penulis :
Ade Maulana
Irmawati
Nanang Durahman
Wirawan Istiono
Johni S Pasaribu
Alexander Waworuntu
Green Ferry Mandias
Raymond Sunardi Oetama
Arief Yanto Rukmana

ISBN :

Editor : Ari Yanto M.Pd
Penyunting : Tri Putri Wahyuni,S.Pd
Desain Sampul dan Tata Letak : Atyka Trianisa, S.Pd

Penerbit : GET PRESS INDONESIA
Anggota IKAPI No. 033/SBA/2022

Redaksi :
Jln. Palarik Air Pacah No 26 Kel. Air Pacah
Kec. Koto Tangah Kota Padang Sumatera Barat
Website : www. getpress.co.id
Email : adm.getpress@gmail.com

Cetakan pertama, Februari 2024

Hak cipta dilindungi undang-undang
Dilarang memperbanyak karya tulis ini dalam bentuk dan dengan cara apapun tanpa izin tertulis dari penerbit.

KATA PENGANTAR

Segala Puji dan syukur atas kehadirat Allah SWT dalam segala kesempatan. Sholawat beriring salam dan doa kita sampaikan kepada Nabi Muhammad SAW. Alhamdulillah atas Rahmat dan Karunia-Nya penulis telah menyelesaikan Buku Rekayasa Kebutuhan Dan Pemodelan Sistem Perangkat Lunak: Konsep, Teori Dan Praktik Dengan UML ini.

Buku ini membahas Dasar-dasar Rekayasa Kebutuhan, Pemodelan Use Case Diagram, Pemodelan Diagram aktivitas, Pemodelan State Machine Diagram, Pemodelan Sequence Diagram, Teknik Pengumpulan Kebutuhan, Perancangan Arsitektur Sistem dengan UML, Verifikasi dan Validasi Kebutuhan dengan UML, Masa Depan Rekayasa Kebutuhan dan Pemodelan.

Proses penulisan buku ini berhasil diselesaikan atas kerjasama tim penulis. Demi kualitas yang lebih baik dan kepuasan para pembaca, saran dan masukan yang membangun dari pembaca sangat kami harapkan.

Penulis ucapkan terima kasih kepada semua pihak yang telah mendukung dalam penyelesaian buku ini. Terutama pihak yang telah membantu terbitnya buku ini dan telah mempercayakan mendorong, dan menginisiasi terbitnya buku ini. Semoga buku ini dapat bermanfaat bagi masyarakat Indonesia.

Padang, Februari 2024

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
DASAR-DASAR REKAYASA KEBUTUHAN

Oleh Ade Maulana

1.1 Pendahuluan

Pada era digital yang terus berkembang, pengembangan perangkat lunak menjadi suatu keharusan untuk memenuhi kebutuhan yang semakin kompleks dari berbagai sektor. Dalam siklus pengembangan perangkat lunak, pemahaman mendalam terhadap kebutuhan adalah kunci untuk menghasilkan solusi perangkat lunak lunak yang memuaskan (Maulana et al., 2023). Kebutuhan yang baik adalah peta jalan yang jelas menuju pencapaian tujuan proyek. Seperti yang diungkapkan oleh Roger S. Pressman (Pressman, 2005), seorang pakar rekayasa perangkat lunak, "Pemahaman yang mendalam terhadap kebutuhan adalah langkah awal yang kritis dalam membangun dasar yang kuat untuk seluruh proyek perangkat lunak."

Memahami kebutuhan dengan baik membantu dalam mengidentifikasi masalah yang sebenarnya dan memastikan bahwa solusi yang dihasilkan benar-benar memenuhi kebutuhan pengguna akhir. Tanpa pemahaman yang memadai terhadap kebutuhan, proyek berisiko mengalami kesulitan dan ketidakpastian, yang dapat berujung pada kegagalan proyek (Sommerville, 2016).

Hubungan antara kebutuhan yang baik dan keberhasilan proyek perangkat lunak bersifat intrinsik. James Cadle dan Donald Yeates (Cadle & Yeates, 2004), dalam buku "Project Management for Information Systems," menekankan, "Kebutuhan yang jelas dan terdokumentasi dengan baik adalah fondasi dari semua keputusan dalam pengembangan perangkat lunak."

Kebutuhan yang baik membantu dalam merinci ekspektasi, mengurangi ketidakpastian, dan memfasilitasi

komunikasi yang efektif antara semua stakeholder. Dengan demikian, kebutuhan yang baik tidak hanya mendukung proses teknis pengembangan, tetapi juga mengarah pada hasil yang memuaskan untuk semua pihak yang terlibat.

1.2 Definisi Kebutuhan

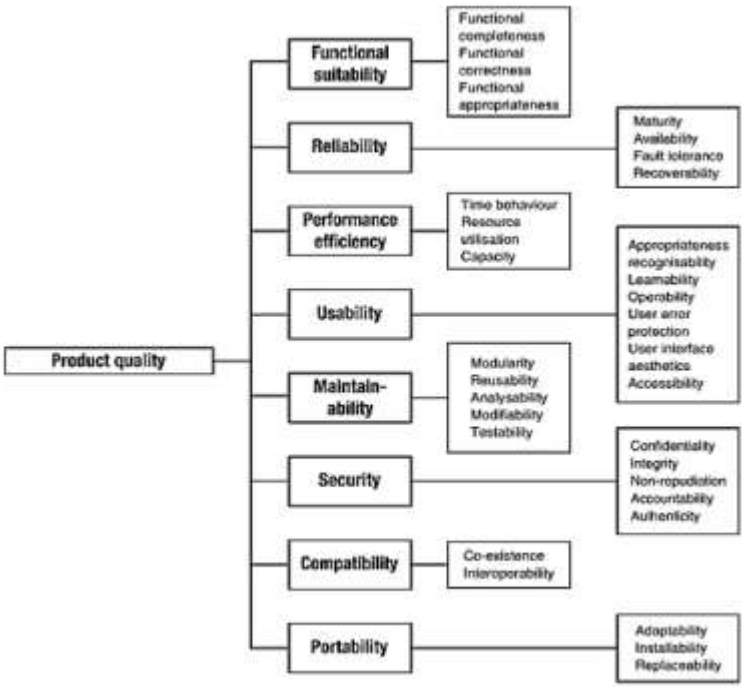
Dalam dunia rekayasa perangkat lunak, konsep kebutuhan merujuk pada kondisi atau kapabilitas yang harus dipenuhi oleh sistem atau komponen perangkat lunak. Pengertian dasar kebutuhan membentuk landasan bagi seluruh siklus pengembangan perangkat lunak, dan pemahaman yang tepat terhadap kebutuhan menjadi kunci untuk mencapai keberhasilan proyek.

Kebutuhan perangkat lunak dibagi menjadi dua kategori utama, yakni kebutuhan fungsional dan non-fungsional. Kebutuhan Fungsional adalah aspek kritis dalam perancangan dan pengembangan sistem perangkat lunak yang menggambarkan perilaku dan fungsi-fungsi spesifik yang diharapkan dari sistem tersebut. Kebutuhan ini secara langsung berkaitan dengan apa yang sistem seharusnya lakukan dan bagaimana pengguna atau komponen lainnya akan berinteraksi dengan sistem tersebut. Dalam konteks pengembangan perangkat lunak, kebutuhan fungsional memberikan panduan yang jelas tentang kemampuan sistem dalam menjalankan tugas-tugas tertentu. Misalnya, kebutuhan fungsional dapat mencakup kemampuan sistem untuk mengelola dan merekam data, memproses transaksi, atau memberikan antarmuka pengguna yang ramah. Keterpenuhan kebutuhan fungsional ini menjadi kunci untuk memastikan bahwa sistem memenuhi harapan pengguna dan memperoleh tingkat kinerja yang diinginkan. Oleh karena itu, identifikasi dan pemahaman yang mendalam terhadap kebutuhan fungsional adalah langkah krusial dalam tahapan perencanaan dan pengembangan sistem perangkat lunak.

Tabel 1.1. Kebutuhan Fungsional Sistem Informasi Stationary (Chairis & Maulana, 2022)

No.	Kategori	Kebutuhan Fungsional
1	Aksesibilitas	Sistem harus dapat berjalan secara online berbasis web
2	Aksesibilitas	Sistem harus berbasis web untuk meminimalisir penggunaan kertas
3	Proses	Diperlukan mekanisme persetujuan online untuk mempercepat proses permintaan
4	Laporan	Admin harus dapat dengan mudah melakukan rekapan data
5	Laporan	Sistem harus memiliki kemampuan memprediksi pengeluaran bulan lalu untuk mempermudah pemesanan stok bulan depan

Kebutuhan Non-Fungsional mencakup aspek-aspek sistem yang tidak secara langsung berkaitan dengan perilaku atau fungsi spesifik sistem, tetapi mempengaruhi kinerja dan penggunaan sistem secara keseluruhan. Kebutuhan ini lebih bersifat kualitatif dan berfokus pada atribut-atribut sistem yang mendukung atau membatasi fungsionalitasnya. Beberapa contoh kebutuhan non-fungsional melibatkan aspek-aspek attribut kualitas perangkat lunak.



Gambar 1.1. Product Quality Model ISO/IEC 25010 (Wattiheluw et al., 2019)

ISO/IEC adalah standar internasional yang umumnya digunakan untuk mengevaluasi dan mengukur kualitas perangkat lunak. Dalam kerangka penelitian ini, dipilihlah ISO/IEC 25010 sebagai standar yang diperbarui dari ISO/IEC 9126, dengan peningkatan struktur dan komponen dari model kualitas standar. ISO/IEC 25010 menggambarkan 8 karakteristik utama yang digunakan untuk menilai secara menyeluruh kualitas perangkat lunak, termasuk portabilitas, efisiensi kinerja, kehandalan, keamanan, kegunaan, kemudahan perawatan, kompatibilitas, dan kesesuaian fungsional. Gambar 1.1 memberikan ilustrasi karakteristik tersebut. Beberapa definisi karakteristik ISO/IEC 25010 melibatkan konsep-konsep seperti berikut:

1. *Functional Suitability* atau Kesesuaian fungsional merujuk pada kemampuan produk aplikasi untuk memberikan

fungsi yang sesuai dengan kebutuhan dalam situasi tertentu.
2. *Reability* atau Kehandalan mengukur sejauh mana produk aplikasi dapat mempertahankan kinerjanya pada tingkat tertentu saat digunakan dalam kondisi tertentu.
3. *Performance efficiency* atau Efisiensi kinerja menilai sejauh mana produk aplikasi memberikan kinerja yang optimal dengan penggunaan sumber daya yang efisien.
4. *Usability* atau Kegunaan melibatkan kemudahan pemahaman, penggunaan, dan daya tarik produk aplikasi bagi pengguna.
5. *Maintainability* atau Kemudahan perawatan menunjukkan sejauh mana produk aplikasi dapat dimodifikasi untuk perbaikan, pengembangan, atau adaptasi sesuai kebutuhan lingkungan.
6. *Security* atau Keamanan menilai tingkat perlindungan yang disediakan oleh produk aplikasi untuk mencegah akses yang tidak sah, modifikasi, atau pengungkapan informasi yang dapat membahayakan.
7. *Compatibility* atau Kompatibilitas menilai kemampuan komponen aplikasi untuk berinteraksi dan bertukar informasi dengan yang lain.
8. *Portability* atau Portabilitas mengacu pada sejauh mana produk aplikasi dapat dipindahkan dari satu lingkungan ke lingkungan lainnya.

Kebutuhan non-fungsional menggambarkan karakteristik sistem yang menciptakan pengalaman pengguna yang memuaskan dan memastikan bahwa sistem dapat diandalkan. Misalnya, kebutuhan kinerja mungkin menetapkan batas waktu respons sistem atau kapasitas penggunaan yang harus dijaga. Kebutuhan keamanan dapat menetapkan langkah-langkah perlindungan data yang wajib diimplementasikan. Pemahaman yang mendalam terhadap kebutuhan non-fungsional memainkan peran penting dalam merancang sistem yang efisien, aman, dan dapat diandalkan untuk memenuhi kebutuhan pengguna dan pemangku kepentingan lainnya. Tabel

2 menunjukkan contoh kebutuhan non-fungsional pada sistem informasi stationary (Chairis & Maulana, 2022).

Tabel 1.2. Kebutuhan Non Fungsional Sistem Informasi Stationary (Chairis & Maulana, 2022)

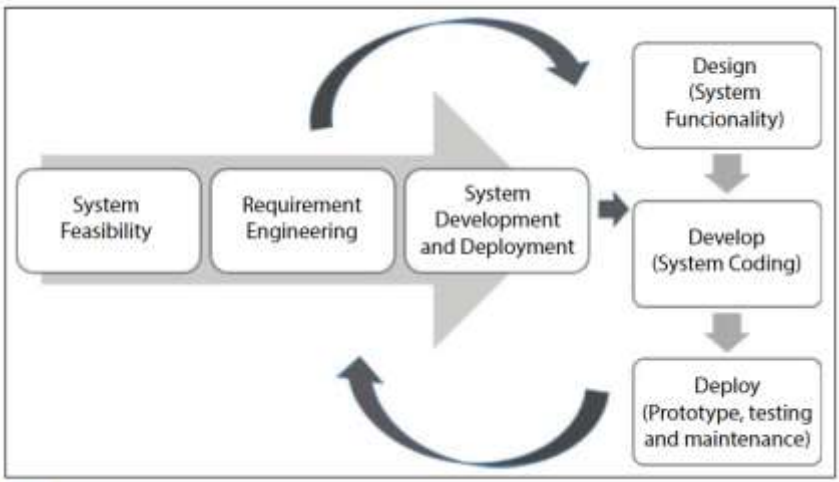
No.	Kategori	Kebutuhan Fungsional
1	Kehandalan	Sistem harus memiliki tingkat ketersediaan yang tinggi (*uptime*) untuk memastikan akses online yang konstan
2	Usabilitas	Antarmuka pengguna harus ramah pengguna dan mudah digunakan
3	Keamanan	Keamanan sistem harus dijaga dengan baik untuk melindungi data pengguna dan informasi perusahaan
4	Efisiensi kinerja	Sistem harus mampu menangani volume data yang besar dengan efisien

1.3 Proses Rekayasa Kebutuhan

Teknik-teknik Rekayasa Kebutuhan telah dibahas dalam literatur dengan tujuan meningkatkan proses penggalian, seperti pemilihan teknik yang sesuai dalam berbagai situasi dan perbandingan efektivitasnya (Anwar & Razali, 2012). Beberapa teknik umum dalam Rekayasa Kebutuhan mencakup Wawancara, Prototipe, Survei, FGD, dan *brainstorming* untuk menangkap kebutuhan pengguna akhir. Penting untuk diingat bahwa satu teknik Rekayasa Kebutuhan tidak dapat mengatasi semua tantangan karena setiap situasi memiliki karakteristik unik yang memerlukan pendekatan berbeda (Tiwari & Rathore, 2017). Oleh karena itu, dianjurkan untuk menggabungkan beberapa teknik penggalian guna mengatasi berbagai jenis tantangan kebutuhan.

Keadaan menjadi sangat kritis jika pengguna akhir tidak mampu menjelaskan kebutuhan sebenarnya selama proses pengumpulan informasi. Dalam banyak kasus, pengguna tidak

dapat mengidentifikasi hubungan antara masalah yang dihadapi dengan kebutuhan sebenarnya yang perlu diatasi. Penggunaan kombinasi teknik Rekayasa Kebutuhan dapat menghasilkan proses penggalian yang melelahkan, yang berpotensi menyebabkan proyek melampaui batas waktu pengiriman produk dan biaya. Dalam skenario semacam ini, proses pengumpulan kebutuhan seharusnya dimulai dengan memahami situasi masalah sebelum menangkap kebutuhan aktual dari klien. Oleh karena itu, dibutuhkan pendekatan alternatif untuk mengeksplorasi skenario tersebut dengan menggunakan kombinasi beberapa teknik Rekayasa Kebutuhan.



Gambar 1.2. Kegiatan Rekayasa Kebutuhan dan Pengembangan Perangkat Lunak (Pawar, 2015)

Tujuan utama dari Rekayasa Kebutuhan adalah menangkap kebutuhan yang sesuai dari pengguna akhir mengenai cara sistem seharusnya berfungsi, dengan harapan untuk meningkatkan efisiensi alur kerja organisasi. Metode yang lebih efektif selama aktivitas kebutuhan dapat menjamin sistem berkualitas, yang tidak hanya hemat biaya tetapi juga meningkatkan kepuasan pelanggan (Prabowo, 2020). Gambar 1.2 menggambarkan keterkaitan penting antara rekayasa kebutuhan dan siklus hidup pengembangan perangkat lunak.

Pada tahap awal, kebutuhan memiliki hubungan yang signifikan dengan kelayakan sistem untuk mengeksplorasi dan memahami bagaimana sistem berfungsi. Selama fase desain dan pengembangan sistem, kebutuhan digunakan sebagai dasar pengambilan keputusan atau aturan untuk mengarahkan tindakan. Terakhir, dalam fase implementasi, tujuannya adalah untuk mengonfirmasi dan memvalidasi fungsionalitas sistem. Menurut (Kumar et al., 2013), prosedur rekayasa kebutuhan dapat dibagi menjadi lima fase utama; Penggalian kebutuhan, Analisis dan Prioritasi kebutuhan, Spesifikasi, Validasi, dan Manajemen Perubahan kebutuhan, sebagaimana ditunjukkan pada Gambar 1.3.

Gambar 1.3. Tahapan pada Rekayasa Kebutuhan

Penggalian dilakukan dalam mode iterasi dan merupakan langkah paling vital untuk menemukan kebutuhan dari para pemangku kepentingan. Proses ini melibatkan penentuan tantangan yang dihadapi oleh pengguna akhir dan merupakan proses untuk menangkap bagaimana sistem baru akan berperforma sesuai dengan kebutuhan pelanggan guna meningkatkan kesulitan operasional. Analisis adalah proses untuk menganalisis semua kebutuhan fungsional dan non-fungsional, serta fase untuk memahami sejauh mana kebutuhan tersebut memenuhi tujuan organisasi. Fase ini akan diselesaikan setelah spesifikasi kebutuhan divalidasi dalam proses validasi kebutuhan. Mengelola perubahan kebutuhan juga sangat penting untuk memastikan bahwa pengiriman proyek tetap sesuai jalur yang benar dan biaya tidak melampaui batas yang telah ditetapkan.

Tugas dalam Rekayasa Kebutuhan memerlukan kegiatan yang ekstensif dari para analis atau insinyur kebutuhan untuk menguraikan sebanyak mungkin pengetahuan taktis menjadi eksplisit, mensintesis pengetahuan yang tersebar, dan

mencocokkan berbagai kepercayaan dan kebutuhan dari para ahli domain. Kurangnya partisipasi pengguna, kurangnya alur kerja atau dokumentasi organisasi yang lengkap, pandangan yang bingung dari pengguna akhir, dan tantangan lainnya sering dihadapi selama proses Rekayasa Kebutuhan. Ini dapat menyebabkan tingkat kegagalan yang tinggi dan peningkatan pekerjaan ulang bug. Oleh karena itu, salah satu solusi kunci untuk mengoptimalkan aktivitas penggalian adalah melalui penerapan teknik Rekayasa Kebutuhan yang sesuai.

1.4 Tantangan dalam Rekayasa Kebutuhan

Rekayasa kebutuhan adalah tahap kritis dalam siklus pengembangan perangkat lunak yang sering kali dihadapkan pada berbagai tantangan. Mengidentifikasi dan mengatasi tantangan ini menjadi kunci untuk mencapai kesuksesan proyek perangkat lunak. Dalam satu penelitian yang dilakukan oleh Tejas Shah, ada 38 masalah dan tantangan yang dalam pengembangan perangkat lunak khususnya rekayasa kebutuhan, Berikut 10 dari 38 masalah dan tantangan yang paling berdampak dalam mengelola kebutuhan (Shah & Patel, 2014):

1. Peningkatan Kualitas Kebutuhan:
   Tantangan utama dalam meningkatkan kualitas kebutuhan adalah merinci kebutuhan secara jelas dan terukur. Solusinya melibatkan inisiasi tugas Quality Assurance (QA) sejak fase Rekayasa kebutuhan, dilanjutkan dengan pelatihan dan kolaborasi antara *requirement engineer*, pemangku kepentingan, dan evaluator untuk memperbaiki kebutuhan.
2. Kurangnya Aktivitas Rekayasa Kebutuhan yang Tersandardisasi:
   Kurangnya aktivitas Rekayasa Kebutuhan yang terstandarisasi dapat menciptakan ketidakpastian dalam proses pengembangan perangkat lunak. Untuk mengatasi tantangan ini, diperlukan penetapan aktivitas Rekayasa Kebutuhan yang terstandarisasi untuk setiap siklus dan rilis

proyek dalam metodologi pengembangan perangkat lunak yang digunakan.

3. Integrasi Model Rekayasa Kebutuhan:
   Mengintegrasikan berbagai model Rekayasa Kebutuhan dalam siklus pengembangan dapat menjadi rumit. Solusinya melibatkan pemahaman dan implementasi model konvensi pemodelan Rekayasa Kebutuhan untuk membantu integrasi berbagai aspek dan elemen kebutuhan.
4. Pemilihan Teknik Elicitation:
   Tantangan dalam memilih teknik elicitation yang sesuai dengan kebutuhan proyek dan karakteristiknya. Solusinya mencakup pengembangan teknik standar yang didedikasikan untuk paradigma tertentu dan pemilihan teknik sesuai dengan konteks proyek.
5. Pandangan yang Bertentangan di Antara Tim :
   Perbedaan pandangan antara tim yang terdistribusi secara global dapat menghambat pemahaman dan koordinasi kebutuhan. Solusinya melibatkan peningkatan komunikasi melalui bahasa teknis yang seragam dan pengaturan pertemuan rutin untuk pemahaman bersama dan pemecahan konflik pandangan.
6. Akurasi dan Pengukuran Kinerja Kebutuhan:
   Mengukur akurasi dan kinerja kebutuhan bisa sulit tanpa matriks pengukuran yang baik. Solusinya melibatkan pengembangan matriks pengukuran akurasi dan performa untuk aktivitas Rekayasa Kebutuhan serta penerapan teknik rekayasa kinerja.
7. Proses Pembuatan Template Umum secara Semi-Otomatis:
   Proses semi-otomatis untuk membuat template umum kebutuhan dapat memerlukan perbaikan dan struktur yang lebih baik. Solusinya mencakup pengembangan proses terstruktur untuk memetakan kebutuhan ke desain secara semi-otomatis dan penggunaan alat-alat Rekayasa Kebutuhan yang mendukung.
8. Dukungan Alat Rekayasa Kebutuhan yang Interaktif:
   Kurangnya dukungan alat Rekayasa Kebutuhan yang interaktif dapat menghambat efisiensi dan efektivitas aktivitas Rekayasa Kebutuhan. Solusinya adalah

menggunakan alat Rekayasa Kebutuhan yang kompatibel, serbaguna, kuat, dan ramah pengguna dengan fasilitas antarmuka elicitation yang efisien.

9. Realisasi Keamanan pada Tingkat Rekayasa Kebutuhan:
   Mewujudkan keamanan pada tingkat Rekayasa Kebutuhan memerlukan perhatian khusus terhadap spesifikasi keamanan tingkat tinggi. Solusinya melibatkan fokus pada spesifikasi keamanan tingkat tinggi dan optimalisasi spesifikasi perilaku ancaman pada tingkat Rekayasa Kebutuhan dan desain.
10. Prioritisasi Kebutuhan:
    Menentukan prioritas kebutuhan untuk memberikan peringkat pada kebutuhan yang berbeda-beda adalah tantangan. Solusinya melibatkan pengembangan strategi identifikasi dan penilaian prioritas untuk memberikan peringkat pada kebutuhan dan memastikan pemenuhan kebutuhan yang paling penting.

Dengan memahami tantangan dan menerapkan strategi yang tepat, tim pengembangan dapat meningkatkan efektivitas rekayasa kebutuhan dan mencapai keberhasilan proyek perangkat lunak.

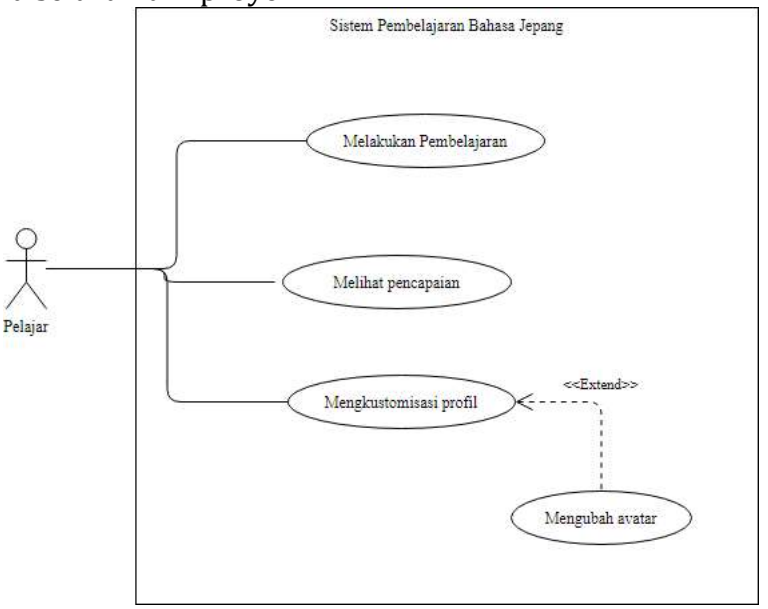
1.5 Peran UML dalam Rekayasa Kebutuhan

Peran Unified Modeling Language (UML) dalam rekayasa kebutuhan memiliki dampak yang signifikan dalam mengembangkan perangkat lunak yang berkualitas. UML, yang merupakan bahasa pemodelan visual standar industri, memberikan kerangka kerja yang jelas dan konsisten untuk mendokumentasikan serta memodelkan kebutuhan perangkat lunak.

Dengan memanfaatkan notasi grafis yang intuitif, UML memungkinkan para pemangku kepentingan, termasuk analis sistem, pengembang perangkat lunak, dan manajemen proyek, untuk berkomunikasi secara efektif (Jacobson & Booch, 2021). Melalui diagram-dokumentasi seperti Use Case Diagrams, Activity Diagrams, Class Diagrams, dan Sequence Diagrams, UML

membantu dalam menangkap dan menggambarkan berbagai aspek kebutuhan, mulai dari skenario penggunaan hingga struktur data yang dibutuhkan.

Sebagai contoh penerapan UML dalam situasi nyata, sekelompok pengembang perangkat lunak dapat menggunakan UML untuk merinci dan memodelkan kebutuhan dari sebuah sistem Informasi Pembelajran Bahasa Jepang yang dapat dilihat pada Gambar 1.4. Use Case Diagram akan mengidentifikasi interaksi antara pengguna dan sistem, Activity Diagram akan memberikan pandangan terhadap alur kerja sistem, sedangkan Class Diagram akan memodelkan entitas seperti Pelajar, Pencapaian, Latihan dan Materi Pelajaran beserta hubungan di antara mereka. Studi kasus semacam ini membuktikan bahwa UML tidak hanya menggambarkan kebutuhan dengan jelas, tetapi juga memberikan dasar yang kokoh untuk pengembangan selanjutnya dengan memastikan pemahaman yang konsisten di antara seluruh tim proyek.



Gambar 1.4. Use Case Diagram Sistem Informasi Pembelajaran Bahasa Jepang (Pangaribuan et al., 2022)

DAFTAR PUSTAKA

Anwar, F., & Razali, R. 2012. A practical guide to requirements elicitation techniques selection-an empirical study. *Middle-East Journal of Scientific Research*, *11*(8), 1059–1067.

Cadle, J., & Yeates, D. 2004. *Project management for information systems*. Pearson education.

Chairis, G., & Maulana, A. 2022. Analisis Perancangan dan Implementasi Sistem Informasi Stationary Berbasis Web pada PT. Indako Trading Coy. *Journal Information System Development (ISD)*, *7*(2), 78–90.

Jacobson, L., & Booch, J. R. G. 2021. *The unified modeling language reference manual*.

Kumar, M., Shukla, M., & Agarwal, S. 2013. A hybrid approach of requirement engineering in agile software development. *2013 International Conference on Machine Intelligence and Research Advancement*, 515–519.

Maulana, A., Heryana, N., Pasaribu, J. S., Aditya, A., Elisawati, Rudiansyah, Amna, Permana, A. A., Rukmana, A. Y., Abdillah, R., & Wahyono, T. 2023. *Rekayasa Perangkat Lunak: Konsep, Metode, dan Praktik Terbaik*. Global Eksekutif Teknologi.

Pangaribuan, J. J., Maulana, A., Zesty, J., Nadjar, F., & others. 2022. Simpuru: Gamifikasi Pembelajaran Bahasa Jepang dalam Aplikasi Berbasis Web. *Jurnal Teknologi Dan Ilmu Komputer Prima (JUTIKOMP)*, *5*(2), 60–65.

Pawar, R. P. 2015. A comparative study of agile software development methodology and traditional waterfall model. *IOSR Journal of Computer Engineering*, *2*(2), 1–8.

Prabowo, M. 2020. *Metodologi Pengembangan Sistem Informasi*. LP2M Press IAIN Salatiga.

Pressman, R. S. 2005. *Software engineering: a practitioner's approach*. Palgrave macmillan.

Shah, T., & Patel, S. V. 2014. A review of requirement engineering issues and challenges in various software development methods. *International Journal of Computer Applications*, *99*(15), 36–45.

Sommerville, I. 2016. *Software engineering*.

Tiwari, S., & Rathore, S. S. 2017. A methodology for the selection of requirement elicitation techniques. *ArXiv Preprint ArXiv:1709.08481*.

Wattiheluw, F. H., Rochimah, S., & Fatichah, C. 2019. Klasifikasi kualitas perangkat lunak berdasarkan iso/iec 25010 menggunakan ahp dan fuzzy mamdani untuk situs web e-commerce. *Jurnal Ilmiah Teknologi Informasi*, *17*(1), 73–83.

BAB 2
PEMODELAN USE CASE DIAGRAM

Oleh Irmawati

2.1 Pendahuluan

Pengembangan perangkat lunak merupakan suatu proses yang kompleks dan memerlukan pemahaman yang baik terhadap kebutuhan pengguna. Dalam konteks ini, *Use Case* Diagram hadir sebagai alat visual yang memfasilitasi pemodelan fungsionalitas sistem dari perspektif pengguna akhir.

Latar belakang dari *Use Case* Diagram mencakup (Fowler, 2018):

1. Pemahaman Kebutuhan Pengguna:
   Use Case Diagram membantu tim pengembang dan stakeholder untuk memahami kebutuhan pengguna dan bagaimana pengguna berinteraksi dengan sistem.
2. Komunikasi yang Efektif:
   Use case diagram memfasilitasi komunikasi antara pengembang, desainer, dan pemangku kepentingan lainnya. Diagram ini menyediakan representasi visual yang jelas tentang fungsionalitas sistem.
3. Pemodelan Fungsionalitas:
   Use Case Diagram membantu dalam pemodelan fungsionalitas sistem dari sudut pandang pengguna, bukan dari implementasi teknis. Hal ini membantu fokus pada apa yang sistem lakukan daripada bagaimana melakukannya.
4. Pengembangan Iteratif:
   Diagram ini dapat digunakan sebagai dasar untuk pengembangan iteratif, memungkinkan perubahan dan penyesuaian kebutuhan pengguna sepanjang waktu.

Tujuan *Use Case* Diagram:

1. *Use Case* Diagram digunakan untuk mengidentifikasi aktor dan *use case*. Aktor adalah entitas yang berinteraksi dengan sistem, sedangkan use case adalah tindakan atau fungsi

yang dilakukan oleh sistem sebagai tanggapan terhadap interaksi tersebut.

2. Diagram ini bertujuan untuk menyajikan secara jelas fungsionalitas sistem dari perspektif pengguna. Ini mencakup tindakan atau skenario penggunaan yang diinginkan.
3. Mendukung spesifikasi fungsionalitas sistem. Dengan memfokuskan pada apa yang dibutuhkan oleh pengguna, dapat membantu dalam mengembangkan pemahaman yang mendalam tentang kebutuhan sistem.
4. *Use Case* Diagram berfungsi sebagai alat komunikasi yang efektif. Dengan menyediakan representasi visual yang dapat dimengerti oleh berbagai pihak, diagram ini memfasilitasi diskusi dan pemahaman bersama.
5. Diagram ini dapat digunakan sebagai dasar untuk analisis, perancangan, dan pengembangan sistem secara keseluruhan.

2.2 Konsep Dasar *Use Case* Diagram

Use Case Diagram adalah alat visual dalam *Unified Modeling Language* (UML) yang digunakan untuk menggambarkan interaksi antara aktor (entitas yang berinteraksi dengan sistem) dan fungsionalitas sistem atau aplikasi. Konsep dasar *Use Case* Diagram melibatkan dua elemen utama: aktor dan use case. Aktor merupakan entitas luar sistem yang berinteraksi dengan sistem, bisa berupa pengguna atau sistem eksternal lainnya. Masing-masing aktor direpresentasikan oleh ikon manusia atau bentuk lain dengan label yang menunjukkan peran atau identitas aktor tersebut.

Use case mewakili fungsi atau tugas tertentu yang dapat dilakukan oleh sistem. Setiap use case direpresentasikan oleh oval dengan label yang deskriptif. Garis hubungan menggambarkan interaksi antara aktor dan use case, menunjukkan bagaimana aktor terlibat dalam fungsionalitas sistem. Selain itu, konsep inklusi dan ekstensi digunakan untuk menunjukkan hubungan antara use case, di mana satu use case dapat mencakup fungsionalitas dari use case lain atau

memperluas fungsionalitas tertentu tanpa menghentikan jalannya *use case* utama.

Garis pembatas sistem mengelilingi aktor dan use case yang ada dalam sistem, menandai batas sistem yang sedang dijelaskan. *Use Case* Diagram membantu dalam memvisualisasikan secara jelas bagaimana pengguna atau aktor eksternal berinteraksi dengan sistem serta fungsionalitas utama yang disediakan oleh sistem tersebut. Diagram ini membantu dalam pemahaman dan komunikasi mengenai kebutuhan fungsional sistem secara keseluruhan.

2.2.1 Komponen Use Case Diagram

Use Case Diagram dalam Unified Modeling Language (UML) terdiri dari beberapa komponen utama yang membantu dalam menggambarkan interaksi antara aktor dan fungsionalitas sistem. Berikut adalah penjelasan singkat tentang komponen-komponen utama dalam Use Case Diagram (Larman, 2012) :

1. Aktor (*Actor*) :
   Aktor adalah entitas luar sistem yang berinteraksi dengan sistem. Ini bisa berupa pengguna, sistem eksternal, atau entitas lain yang memiliki peran tertentu terkait dengan sistem.
   Dalam diagram, aktor direpresentasikan oleh ikon manusia atau bentuk lain dengan label yang menunjukkan peran atau identitas aktor tersebut.
2. *Use Case* :
   Use Case adalah fungsi atau tugas spesifik yang dapat dilakukan oleh sistem. *Use Case* merepresentasikan suatu skenario yang menggambarkan interaksi antara aktor dan sistem.
   Dalam diagram, use case direpresentasikan oleh oval dengan label yang deskriptif, menjelaskan fungsi atau tugas yang dilakukan oleh sistem.
3. Relasi Aktor-*Use Case* :
   Garis hubungan menghubungkan aktor dengan use case, menunjukkan keterlibatan aktor dalam suatu fungsi atau

tugas tertentu. Hubungan ini mencerminkan bagaimana aktor berinteraksi dengan fungsionalitas sistem.
Hubungan ini dapat berupa garis lurus atau panah yang mengarah dari aktor ke use case.

4. *Include* (Inklusi) dan *Extend* (Ekstensi) :
   Include menunjukkan bahwa satu *use case* memasukkan (menggunakan) fungsionalitas dari *use case* lain. Menunjukkan hubungan inklusi antara dua use case.
   Extend menunjukkan bahwa satu *use case* dapat diperluas dengan fungsionalitas tambahan dari *use case* lain. Menunjukkan hubungan ekstensi antara dua *use case*.
5. *Sistem Boundary* :
   Garis pembatas sistem mengelilingi aktor dan use case yang ada dalam sistem. Menandai batas sistem yang sedang dijelaskan dalam diagram.

Semua komponen tersebut bekerja bersama untuk memberikan pemahaman yang jelas tentang interaksi antara pengguna atau aktor eksternal dengan sistem, serta fungsionalitas utama yang disediakan oleh sistem tersebut. Use Case Diagram memberikan pandangan tingkat tinggi tentang kebutuhan fungsional sistem dan dapat digunakan sebagai dasar untuk pengembangan lebih lanjut dalam tahap analisis dan desain.

2.3 Pemodelan Use Case Diagram

Pemodelan *Use Case* Diagram merupakan salah satu tahap dalam proses pengembangan perangkat lunak yang menggunakan teknik pemodelan berorientasi objek. Use Case Diagram digunakan untuk menggambarkan interaksi antara sistem dengan entitas atau aktor yang berinteraksi dengannya. *Use Case* sendiri merupakan representasi dari fungsionalitas atau tindakan yang dapat dilakukan oleh sistem. Diagram ini membantu dalam memahami dan menggambarkan bagaimana sistem berinteraksi dengan aktor-aktor eksternal, serta menunjukkan fungsionalitas utama yang disediakan oleh sistem.

Berikut adalah langkah-langkah umum untuk memodelkan Use Case Diagram:

1. Identifikasi Aktor:
   Tentukan siapa atau apa yang berinteraksi dengan sistem. Identifikasi aktor-aktor eksternal yang memiliki peran dalam menggunakan sistem.
2. Identifikasi Use Case:
   Identifikasi fungsionalitas atau tindakan yang dapat dilakukan oleh sistem. Fokus pada aktivitas utama yang dilakukan oleh pengguna atau sistem eksternal.
3. Tentukan Hubungan Aktor-Use Case:
   Tentukan hubungan antara aktor dan use case. Aktor mana yang terlibat dalam setiap use case dan apa peran mereka.
4. Gambarkan Use Case Diagram:
   Gunakan notasi UML untuk menggambarkan aktor, use case, dan hubungan diantara mereka. Aktor direpresentasikan dengan ikon manusia atau sesuatu yang sesuai dengan peran mereka.
5. Definisikan *Boundary* (Batas):
   Tentukan batas sistem dengan menggambar persegi panjang di sekitar use case. Membatasi area di mana use case dan aktor berada.
6. Tentukan Relasi Inklusi dan Ekstensi (Optional):
   Jika diperlukan, tentukan relasi inklusi *(Include*) dan ekstensi (*Extend*) antara use case. Membantu menggambarkan alur logika tambahan atau ekstensi fungsionalitas.
7. Berikan Nama dan Deskripsi:
   Berikan nama yang jelas dan deskripsi singkat untuk setiap aktor dan use case. Membantu dalam pemahaman yang lebih baik tentang tujuan dan peran masing-masing.
8. Validasi dengan Pemangku Kepentingan:
   Validasi diagram dengan pemangku kepentingan seperti pengguna, pengembang, dan pemilik produk. Pastikan bahwa use case diagram mencerminkan kebutuhan bisnis yang sebenarnya.

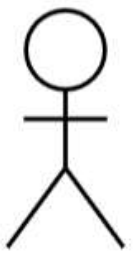
2.3.1 Identifikasi Aktor

Identifikasi aktor dalam UML merupakan tahap awal dalam pemodelan sistem, dimana entitas luar yang berinteraksi dengan sistem diidentifikasi sebagai aktor. Aktor ini merepresentasikan peran yang dapat dimainkan oleh individu, perangkat lunak, atau perangkat keras eksternal. Dengan mengidentifikasi aktor, kita dapat memahami bagaimana interaksi antara sistem dan entitas luar terjadi.

Contoh identifikasi aktor meliputi (Eriksson *et al.*, 2003):

1. Pelanggan: Aktor ini mewakili individu atau organisasi yang menggunakan sistem untuk melakukan pembelian produk atau layanan.
2. Administrator: Aktor ini mencakup pengguna yang memiliki hak akses tinggi dan dapat mengelola atau mengkonfigurasi sistem.
3. Sistem Eksternal: Dapat mencakup integrasi dengan sistem eksternal seperti sistem pembayaran pihak ketiga atau layanan pihak ketiga.
4. Pengunjung Situs Web: Jika sistem yang dimodelkan adalah situs web, pengunjung situs web dapat dianggap sebagai aktor.

Notasi aktor dalam UML digambarkan sebagai simbol manusia atau bentuk lain yang merepresentasikan entitas luar yang berinteraksi dengan sistem. Aktor biasanya ditempatkan disebelah kiri atau kanan diagram, yang mewakili entitas luar yang terlibat dalam interaksi dengan sistem.



Gambar 2.1. Notasi aktor

Saat memodelkan sistem dengan UML, lebih dari satu aktor dapat diidentifikasi, dan setiap aktor dapat memiliki peran atau tanggung jawab yang berbeda terhadap sistem. Identifikasi aktor membantu dalam merinci kebutuhan dan skenario penggunaan sistem yang diperlukan.

Dalam konteks UML, aktor adalah entitas luar yang berinteraksi dengan sistem. Aktor-aktor ini dapat dibagi menjadi beberapa jenis berdasarkan perannya dan hubungannya dengan sistem. Berikut adalah beberapa jenis aktor umum:

1. *Primary Actor* (Aktor Utama):
   Aktor utama adalah entitas luar yang memiliki tujuan utama atau kepentingan langsung dalam mencapai fungsi utama sistem.
   Contoh: Jika memodelkan sistem pemesanan online, pelanggan adalah aktor utama karena mereka memiliki tujuan utama untuk memesan produk atau layanan.
2. *Secondary Actor* (Aktor Sekunder):
   Aktor sekunder adalah entitas luar yang berinteraksi dengan sistem, namun, kepentingan dan tujuannya tidak sebesar aktor utama.
   Contoh: Dalam sistem pemesanan online, pihak kurir atau sistem pembayaran pihak ketiga dapat dianggap sebagai aktor sekunder.
3. *Boundary Class Actor* (Aktor Kelas Batasan):
   Aktor ini mewakili batasan atau batas sistem dan merupakan titik masuk atau keluar dari sistem.
   Contoh: Dalam sistem perpustakaan, pustakawan dapat dianggap sebagai aktor batasan karena mereka berada di batas sistem dan berinteraksi dengan pelanggan serta sistem perpustakaan.
4. *Human Actor* (Aktor Manusia):
   Aktor manusia adalah individu atau kelompok orang yang berinteraksi dengan sistem.
   Contoh: Pengguna, administrator, atau pelanggan adalah contoh aktor manusia.

5. *System Actor* (Aktor Sistem):
   Aktor sistem adalah komponen perangkat keras atau perangkat lunak yang berinteraksi dengan sistem.
   Contoh: Sensor, database, atau sistem eksternal dapat dianggap sebagai aktor sistem.
6. Temporal Actor (Aktor Temporal):
   Aktor temporal adalah entitas luar yang berinteraksi dengan sistem dalam konteks waktu tertentu atau kejadian tertentu.
   Contoh: Aktor yang muncul hanya pada waktu tertentu, seperti "hari libur nasional" atau "periode promosi."

Penting untuk mengidentifikasi dan menggambarkan berbagai jenis aktor ini saat memodelkan sistem, karena dapat membantu dalam pemahaman yang lebih baik tentang cara sistem berinteraksi dengan dunia luar.

Peran aktor dalam konteks UML merujuk pada tanggung jawab atau fungsi khusus yang diemban oleh aktor tersebut dalam interaksi dengan sistem. Identifikasi peran aktor membantu dalam merinci bagaimana setiap entitas luar berkontribusi dalam mencapai tujuan sistem. Berikut adalah beberapa peran umum yang dapat dimainkan oleh aktor:

Peran aktor dalam konteks Unified Modeling Language (UML) merujuk pada tanggung jawab atau fungsi khusus yang diemban oleh aktor tersebut dalam interaksi dengan sistem. Identifikasi peran aktor membantu dalam merinci bagaimana setiap entitas luar berkontribusi dalam mencapai tujuan sistem. Berikut adalah beberapa peran umum yang dapat dimainkan oleh aktor:

1. Inisiator (*Initiator*): Memulai atau memicu interaksi dengan sistem.
   Contoh: Pelanggan yang memulai proses pemesanan produk atau layanan.
2. Pengguna (*User*): Menggunakan fungsionalitas sistem untuk mencapai tujuan tertentu.
   Contoh: Pengguna yang menggunakan antarmuka pengguna untuk berinteraksi dengan sistem.

3. Administrator: Bertanggung jawab untuk mengelola, mengawasi, atau mengkonfigurasi sistem.
   Contoh: Administrator sistem yang memiliki hak akses tinggi untuk mengelola pengguna dan konfigurasi sistem.
4. Penyedia Layanan (*Service Provider*): Menyediakan layanan atau informasi kepada sistem.
   Contoh: Sistem pembayaran pihak ketiga yang menyediakan layanan pembayaran.
5. Kolaborator (*Collaborator*): Berkolaborasi atau berinteraksi dengan sistem untuk mencapai tujuan bersama.
   Contoh: Pihak ketiga yang terlibat dalam proses bisnis tertentu, seperti mitra bisnis atau penyedia logistik.
6. Sistem Eksternal (*External System*): Berinteraksi dengan sistem sebagai entitas eksternal.
   Contoh: Integrasi dengan sistem lain, seperti sistem inventaris eksternal atau sistem pelacakan pengiriman.
7. Penerima Informasi (*Information Receiver*): Menerima informasi dari sistem.
   Contoh: Penerima informasi seperti pelanggan yang menerima notifikasi atau konfirmasi dari sistem.
8. Analisis (*Analyzer*): Menganalisis data atau informasi yang dihasilkan oleh sistem.
   Contoh: Analis data yang menggunakan laporan sistem untuk mengambil keputusan.
9. Pemeriksa (*Inspector*): Memeriksa atau mengawasi proses atau hasil dari sistem.
   Contoh: Auditor yang memeriksa kepatuhan sistem terhadap standar atau regulasi tertentu.

Setiap aktor dapat memiliki beberapa peran, dan identifikasi peran membantu dalam merinci interaksi antara aktor dan sistem. Pemahaman yang baik tentang peran aktor membantu dalam merancang sistem yang responsif dan sesuai dengan kebutuhan pengguna dan pemangku kepentingan.

2.3.2 Identifikasi Use Case

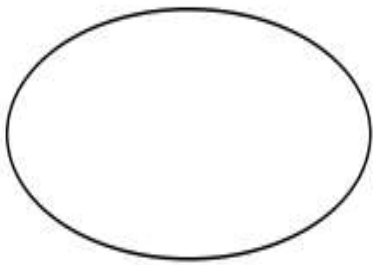
Identifikasi use case adalah proses pengenalan, penentuan, dan penjelasan dari fungsionalitas-fungsionalitas utama atau tindakan-tindakan yang dapat dilakukan oleh sebuah sistem. Use case mendeskripsikan interaksi antara aktor (pengguna atau sistem eksternal) dengan sistem untuk mencapai suatu tujuan tertentu. Proses identifikasi use case melibatkan beberapa langkah kunci:

1. Identifikasi Aktor: Tentukan siapa atau apa yang berinteraksi dengan sistem. Aktor dapat berupa pengguna, sistem eksternal, atau entitas lain yang berpartisipasi dalam interaksi dengan sistem.
2. Identifikasi Tujuan: Identifikasi tujuan atau hasil yang diinginkan dari interaksi antara aktor dan sistem. Tujuan ini akan menjadi dasar untuk menentukan use case.
3. Pemetaan Skenario: Gambaran secara rinci tentang bagaimana aktor berinteraksi dengan sistem untuk mencapai tujuan tertentu. Identifikasi langkah-langkah atau proses yang diperlukan untuk mencapai tujuan tersebut.
4. Menentukan Use Case: Tentukan fungsionalitas atau tindakan-tindakan yang membentuk use case. Setiap use case harus memiliki tujuan yang jelas dan dapat diidentifikasi.
5. Penentuan Ketergantungan: Tentukan ketergantungan antar use case. Misalnya, apakah suatu use case termasuk ke dalam use case lain atau apakah ada ketergantungan waktu antara use case tertentu.
6. Verifikasi dengan Pemangku Kepentingan: Melibatkan pemangku kepentingan (*stakeholders*) dalam proses identifikasi use case untuk memastikan bahwa semua fungsionalitas utama yang diinginkan telah dicakup.

Proses identifikasi use case ini membantu dalam memahami kebutuhan sistem dan membentuk dasar untuk pengembangan selanjutnya, termasuk pembuatan diagram use case UML.

Notasi use case digunakan dalam UML untuk menggambarkan interaksi antara pengguna atau aktor dengan

sistem. Use case direpresentasikan oleh oval dengan nama use case di dalamnya. Use case menggambarkan fungsionalitas atau tindakan yang dapat dilakukan oleh aktor dalam sistem.



Gambar 2.2. Notasi use case

2.3.3 Menggambar Use Case Diagram

Use Case Diagram adalah diagram yang menggambarkan interaksi antara aktor (yang bisa berupa pengguna atau sistem lain) dengan suatu sistem untuk mencapai suatu tujuan tertentu. Diagram ini merupakan salah satu bagian dari UML, yang digunakan untuk memodelkan dan mendokumentasikan analisis dan desain perangkat lunak.

Notasi pada *Use Case* Diagram menggunakan simbol-simbol tertentu untuk merepresentasikan elemen-elemen seperti aktor, *use case*, dan hubungan di antara mereka. Berikut adalah notasi-notasi yang umum digunakan pada Use Case Diagram (von der Maßen and Lichter, 2002):

1. Aktor (*Actor*):
   Aktor direpresentasikan oleh simbol manusia atau simbol lain yang menunjukkan entitas eksternal yang berinteraksi dengan sistem. Aktor ditempatkan di luar lingkaran use case.
2. *Use Case: Use case* direpresentasikan oleh oval atau elips dan berisi nama dari suatu tindakan atau skenario yang ingin dicapai oleh aktor.
3. Hubungan Aktor dengan *Use Case*:
   Hubungan antara aktor dan use case diilustrasikan dengan garis. Garis ini menunjukkan keterlibatan aktor dalam suatu use case.

4. Relasi antar Use Case:
   Beberapa use case dapat memiliki hubungan khusus di antara mereka, seperti inklusi (*include*), generalisasi (*extend*), atau ekstensi (*extend*). Ini diilustrasikan dengan panah atau garis dengan anotasi khusus.
5. *Boundary* (Batas):
   Batas sistem kadang-kadang ditambahkan untuk menunjukkan konteks atau lingkup dari sistem yang digambarkan. Ini adalah suatu kotak yang mengelilingi use case dan aktor.
6. *Include* dan *Extend* (Inklusi dan Ekstensi):
   Inklusi dan ekstensi adalah jenis relasi antar use case. Include menunjukkan bahwa suatu use case mencakup fungsi dari use case lain. Extend menunjukkan bahwa suatu use case dapat memperluas atau memperpanjang use case lain dalam situasi tertentu.

Use Case Diagram adalah alat visualisasi tingkat tinggi dan digunakan untuk memahami interaksi antara aktor dan sistem. Simbol-simbol di atas membantu menggambarkan struktur dan hubungan dalam cara yang mudah dipahami oleh pemangku kepentingan proyek.

2.4 Hubungan Antar Use Case

Hubungan antar *Use Case* merujuk pada cara *Use Case-Use Case* dalam analisis perangkat lunak saling berinteraksi dan bekerja sama untuk mencapai tujuan tertentu dalam suatu sistem. *Use Case* merupakan teknik dalam analisis perangkat lunak yang digunakan untuk mendokumentasikan fungsionalitas sistem dengan cara mendeskripsikan interaksi antara aktor (pengguna atau sistem eksternal lain) dengan sistem.

Berikut adalah beberapa konsep yang terkait dengan hubungan antar *Use Case* (Jacobson and Booch, 2021):

1. *Include Relationship* (Hubungan Include): Hubungan ini menunjukkan bahwa satu *Use Case* mencakup fungsionalitas dari *Use Case* lainnya. Dengan kata lain, Use *Case* yang

"diinclude" dapat digunakan atau dijalankan oleh *Use Case* yang "menginclude".
Contoh, *Use Case* "*Withdraw Cash*" bisa mencakup *Use Case* "*Authenticate User*".

2. Extend Relationship (Hubungan *Extend*): Hubungan ini menunjukkan bahwa suatu Use Case dapat memperluas atau memperpanjang fungsionalitas *Use Case* lain pada kondisi tertentu. *Use Case* yang "meng-extend" menawarkan fungsionalitas tambahan ketika kondisi tertentu terpenuhi.
Contoh, *Use Case* "*Make Payment*" dapat di-*extend* oleh *Use Case* "*Apply Discount*" pada kondisi tertentu.
3. Generalization Relationship (Hubungan Generalisasi): Mirip dengan konsep pewarisan dalam pemrograman berorientasi objek, menunjukkan bahwa suatu *Use Case* umum dapat diperluas oleh *Use Case* yang lebih khusus. Membantu dalam merangkum fungsionalitas yang serupa dalam hierarki *Use Case*.

2.4.1 Hubungan Include

Dalam diagram *use case*, hubungan "*include*" menggambarkan sebuah ketergantungan di antara *use case-use case* tertentu. Hubungan ini menunjukkan bahwa satu *use case* (yang disebut sebagai use case yang di-"*include")* memerlukan fungsi atau perilaku dari *use case* lain (yang disebut sebagai use case yang "*included*"). Dengan kata lain, use case yang di-"*include*" adalah bagian dari use case yang di-"including".
Berikut adalah beberapa poin kunci tentang hubungan "include" dalam use case diagram:

1. Penggunaan Kembali Fungsionalitas: Hubungan "include" memungkinkan penggunaan kembali fungsionalitas yang sudah ada dalam suatu use case tanpa perlu memodelkan ulang di use case yang lain. Use case yang di-"included" menyediakan suatu fitur atau fungsionalitas yang dapat digunakan oleh use case yang di-"including".
2. Aliran Kontrol: Ketika use case yang di-"including" dijalankan, use case yang di-"included" akan dieksekusi pada suatu titik tertentu. Ini menciptakan suatu aliran kontrol di antara use case-use case tersebut.

3. Penggunaan Saat Eksekusi: Use case yang di-"*included*" tidak berdiri sendiri, dan eksekusinya tergantung pada *use case* yang di-"*including*". Artinya, *use case* yang di-"included" tidak dapat dijalankan secara independen.
4. Hubungan Korelasi: Hubungan "*include*" biasanya mengindikasikan bahwa use case yang di-"including" memerlukan fungsionalitas tambahan dari *use case* yang di-"included" untuk mencapai suatu tujuan tertentu.

Contoh sederhana hubungan "*include*" dapat dilihat dalam suatu sistem pemesanan tiket online. Misalnya, terdapat *use case* "Pesan Tiket" yang dapat mencakup *use case* "Pilih Kursi". Meskipun "Pesan Tiket" dapat dilakukan tanpa harus memilih kursi terlebih dahulu, namun pemilihan kursi menjadi suatu langkah yang di-"*included*" dalam *use case* "Pesan Tiket".

Dalam diagram *use case*, hubungan "include" direpresentasikan dengan garis panah dengan tanda panah menunjuk dari use case yang di-"including" ke *use case* yang di-"*included*". Pada garis panah tersebut, biasanya diberikan label yang menjelaskan kondisi atau situasi ketika *use case* yang di-"*included*" akan dieksekusi oleh *use case* yang di-"including".

2.4.2 Hubungan Extend

Dalam *use case* diagram, hubungan "*extend*" adalah salah satu jenis hubungan antar *use case* yang digunakan untuk menunjukkan bahwa satu *use case* (yang disebut *use case* dasar) dapat memperluas atau memperpanjang fungsionalitas dari *use case* lainnya (yang disebut *use case* ekstensi). Hubungan "extend" digunakan ketika suatu *use case* (*use case* ekstensi) hanya terjadi dalam situasi tertentu yang melibatkan *use case* dasar.

Berikut adalah beberapa poin kunci tentang hubungan "*extend*":

1. *Use Case* Dasar (*Base Use Case*): use case yang memiliki fungsionalitas dasar atau inti. Use case dasar selalu terjadi, tanpa memandang keadaan.
2. *Use Case* Ekstensi (*Extension Use Case*): *use case* ini menyediakan fungsionalitas tambahan yang hanya terjadi

dalam situasi tertentu. Use case ekstensi memperluas atau memperpanjang use case dasar.

3. Trigger: Use case ekstensi dipicu oleh suatu kondisi atau peristiwa tertentu yang terjadi selama eksekusi use case dasar. Trigger ini mengaktifkan atau memicu use case ekstensi untuk berjalan.
4. Urutan Eksekusi: Use case ekstensi dieksekusi setelah use case dasar dan hanya jika kondisi tertentu terpenuhi. Ini memberikan fleksibilitas dan modularitas pada sistem, karena fungsionalitas tambahan dapat ditambahkan tanpa mengubah use case dasar.
5. Notasi Extend: Dalam diagram use case, hubungan extend direpresentasikan dengan panah dengan ujung terbuka yang menunjuk dari *use case* ekstensi ke use case dasar. Terdapat label "*extends*" yang menunjukkan hubungan ini.

Contoh sederhana dapat membantu menjelaskan konsep ini. Misalnya, pertimbangkan *use case* dasar "Pesan Tiket" dan *use case* ekstensi "Tambahkan Layanan Asuransi". Use case ekstensi ini hanya terjadi jika pengguna memilih untuk menambahkan layanan asuransi selama proses pemesanan tiket. Dalam diagram use case, hubungan "*extend*" akan menunjukkan bahwa "Tambahkan Layanan Asuransi" memperluas fungsionalitas dari use case dasar "Pesan Tiket".

2.4.3 Hubungan Generalization

Hubungan generalisasi adalah jenis hubungan yang digunakan untuk menunjukkan hubungan hierarki antara dua entitas. Entitas yang satu (yang lebih umum atau abstrak) disebut sebagai "s*uperclass*" atau "*parent class*," sedangkan yang lainnya (yang lebih spesifik) disebut sebagai "*subclass*" atau "*child class*." Hubungan ini menunjukkan bahwa subclass mewarisi sifat atau perilaku dari *superclass*.

Berikut adalah beberapa poin kunci tentang hubungan generalisasi:

1. *Superclass* (*Parent Class*): entitas yang lebih umum atau abstrak yang berisi sifat-sifat atau perilaku yang bersama-sama dengan satu atau lebih *subclass. Superclass* seringkali

memiliki deskripsi umum dan mungkin tidak diinstansiasi sendiri.
2. *Subclass* (*Child Class*): entitas yang lebih spesifik yang mewarisi sifat-sifat atau perilaku dari superclass. Subclass dapat menambahkan atau memodifikasi sifat-sifat yang diwarisi.
3. *Inheritance* (Pewarisan): Hubungan generalisasi mewakili pewarisan, di mana subclass mewarisi atribut dan metode dari superclass. Ini memungkinkan untuk mendefinisikan hierarki dalam model.
4. Notasi *Generalization*: Dalam diagram use case atau diagram kelas UML, hubungan generalisasi direpresentasikan dengan garis panah yang menunjuk dari subclass ke superclass. Panah umumnya memiliki ujung terbuka pada sisi superclass dan ujung yang tertutup pada sisi subclass.

Contoh sederhana hubungan generalisasi dalam use case diagram adalah *use case* "Pesan Tiket" sebagai superclass dan use case "Pesan Tiket Penerbangan" serta "Pesan Tiket Kereta" sebagai subclass. Use case "Pesan Tiket" mungkin memiliki atribut dan perilaku umum, sedangkan *subclass* masing-masing menambahkan detil khusus untuk pesanan tiket penerbangan dan kereta.

2.5 Teknik Validasi *Use Case* Diagram

Teknik validasi *Use Case* Diagram melibatkan pengecekan dan evaluasi untuk memastikan bahwa diagram tersebut memenuhi kriteria kelayakan dan kesesuaian dengan kebutuhan proyek atau sistem yang sedang dibangun. Beberapa teknik validasi *Use Case* Diagram yang umum digunakan melibatkan:

1. Konsistensi dan Kesesuaian:
   a. Memastikan bahwa setiap use case mencerminkan kebutuhan bisnis atau fungsionalitas sistem.
   b. Menjamin konsistensi antara Use Case Diagram dengan dokumen kebutuhan dan spesifikasi lainnya.

2. Kelengkapan Use Case:
    a. Memastikan bahwa semua aksi atau skenario yang mungkin terjadi telah dicakup oleh use case.
    b. Mengidentifikasi dan menutupi semua aktor yang terlibat dalam sistem.
3. Hubungan Aktor dan Use Case:
    a. Memastikan bahwa hubungan antara aktor dan use case relevan dan sesuai dengan fungsionalitas sistem.
    b. Menilai apakah aktor memiliki akses yang diperlukan untuk menjalankan use case tertentu.
4. Pengelompokan dan Generalisasi:
    a. Mengevaluasi apakah use case yang terkelompok atau di-generalisasi telah dirancang dengan benar.
    b. Memastikan bahwa pewarisan atau pengelompokan use case sesuai dengan hierarki yang diinginkan.
5. Skalabilitas dan Pemeliharaan:
    a. Memeriksa apakah Use Case Diagram dapat berkembang dengan baik seiring waktu dan tetap mudah dimengerti.
    b. Menilai sejauh mana diagram dapat memfasilitasi pemeliharaan dan perubahan pada sistem.
6. Standar dan Pedoman:
    a. Memastikan bahwa diagram mematuhi standar dan pedoman pengembangan perangkat lunak yang berlaku.
7. Uji Coba dan Umpan Balik:
    a. Melibatkan pemanggilan aktor dan use case dalam skenario uji coba untuk memastikan bahwa mereka berperilaku seperti yang diinginkan.
    b. Mengumpulkan umpan balik dari pemanggilan use case kepada pengembang atau pemanggilan aktor untuk memastikan kesesuaian dengan kebutuhan.
8. Review oleh Pihak Terkait:
    a. Melibatkan pihak terkait, seperti pengguna akhir, dalam proses validasi untuk mendapatkan sudut pandang eksternal dan memastikan pemahaman yang benar tentang kebutuhan bisnis.

Teknik validasi dapat bervariasi tergantung pada metodologi pengembangan perangkat lunak yang digunakan dan konteks proyek. Menerapkan teknik validasi yang sesuai dapat membantu memastikan bahwa Use Case Diagram benar-benar mencerminkan kebutuhan dan spesifikasi yang diinginkan.

Dalam validasi, kita akan memeriksa konsistensi antara use case tersebut, menilai apakah semua langkah proses telah dicakup, dan memastikan bahwa setiap aktor memiliki akses yang sesuai dengan tugas mereka. Validasi juga dapat melibatkan uji coba simulasi untuk memastikan bahwa sistem berperilaku sesuai dengan yang dijelaskan dalam Use Case Diagram.

Penting untuk melibatkan pihak terkait, seperti pelanggan atau manajer restoran, dalam proses validasi untuk mendapatkan masukan langsung dan memastikan bahwa Use Case Diagram mencerminkan kebutuhan bisnis secara akurat.

2.6 Tantangan Pemodelan *Use Case* Diagram

Meskipun *Use Case* Diagrams memberikan gambaran yang cukup jelas tentang fungsionalitas sistem, ada beberapa tantangan yang mungkin dihadapi dalam pemodelannya:

1. Ketidakjelasan dalam Identifikasi Use Case:
   Identifikasi use case yang tepat bisa menjadi tantangan. Terkadang, mungkin sulit untuk memutuskan apakah suatu fitur atau fungsionalitas harus diwakili sebagai use case terpisah atau sebagai bagian dari use case yang lebih besar.
2. Tingkat Abstraksi yang Tepat:
   Menentukan tingkat abstraksi yang tepat dalam menggambarkan use case seringkali dapat menjadi sulit. Terlalu rinci bisa membuat diagram terlalu kompleks, jika terlalu umum bisa menyebabkan kehilangan detail yang penting.
3. Komunikasi dengan Stakeholder:
   Menyusun Use Case Diagram yang dapat dipahami dan relevan untuk semua pemangku kepentingan (*stakeholders*) bisa menjadi tantangan. Beberapa orang mungkin memiliki

pemahaman teknis yang lebih tinggi daripada yang lain, sehingga perlu ditemukan cara untuk menjelaskan konsep secara efektif kepada semua pihak.

4. Perubahan Kebutuhan:
   Kebutuhan bisnis dan fungsionalitas sistem bisa berubah seiring waktu. Merespon perubahan ini dan memperbarui Use Case Diagram secara konsisten dapat menjadi tugas yang menantang.
5. Pertimbangan Performa:
   Use Case Diagrams cenderung lebih fokus pada fungsionalitas daripada performa sistem. Untuk proyek dengan persyaratan kinerja yang ketat, perlu dikombinasikan dengan diagram lain, seperti diagram sekuen atau diagram aktivitas, untuk mencakup aspek-aspek kinerja.
6. Keterbatasan dalam Menangkap Interaksi yang Kompleks:
   Use Case Diagrams tidak selalu mampu menangkap dengan baik interaksi yang sangat kompleks atau detail tinggi antara use case. Untuk kasus seperti itu, mungkin diperlukan diagram-diagram tambahan atau metode pemodelan yang lebih rinci.
7. Pemodelan Kesalahan dan Eksepsi:
   Menggambarkan bagaimana sistem menangani kesalahan dan eksepsi dapat menjadi tantangan. Dalam beberapa kasus, use case diagram tidak menyediakan representasi yang cukup baik untuk menangkap skenario kesalahan.

Saat menggunakan Use Case Diagrams, penting untuk diingat bahwa ini hanya salah satu alat dalam kotak perang pemodelan, dan seringkali perlu digunakan bersama dengan diagram UML lainnya untuk memberikan gambaran yang lengkap dan akurat tentang sistem yang direncanakan atau yang ada.

DAFTAR PUSTAKA

Eriksson, H.-E. *et al.* 2003. *UML 2 toolkit*. John Wiley & Sons.

Fowler, M. 2018. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional.

Jacobson, L. and Booch, J.R.G. 2021. 'The unified modeling language reference manual'.

Larman, C. 2012. *Applying UML and patterns: an introduction to object oriented analysis and design and interative development*. Pearson Education India.

von der Maßen, T. and Lichter, H. 2002. 'Modeling variability by UML use case diagrams', in *Proceedings of the International Workshop on Requirements Engineering for product lines*, pp. 19–25.

BAB 3
PEMODELAN DIAGRAM AKTIVITAS

Oleh Nanang Durahman

3.1 Pendahuluan

Activity Diagram adalah sebuah diagram UML yang menggambarkan deskripsi aliran kerja dalam proses bisnis secara dinamis dari sistem. Diagram tersebut hampir sama dengan flowchart pada perancangan struktur perbedaanya diagram aktivitas mendukung perilaku paralel yang mungkin terjadi pada beberapa eksekusi sedangkan flowchart atau Data Flow Diagram (Primadewi dan Nugroho, 2022). Dalam digaram aktivitas digambarkan berdasarkan diagram usecase yang terdiri dari berbagai aliran aktivitas, langkah-langkah atau urutan aktivitas dalam sistem yang sedang dirancang, mulai dari mana berawal dan berakhir, percabangan yang memungkinkan terjadi dan bagaimana aktivitas berakhir. Aktivitas tersebut menggambarkan sebuah proses yang berjalan, sementara use case menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

Diagram dapat menguntungkan bagi pembaca sistem karena membantu membaca dan memahami seluru proses aliran sistem dalam beberapa use case. (Melisa dan Agung, 2019), karena Diagram aktivitas hanya menggambarkan aktivitas dari sistem bukan aktivitas yang dilakukan oleh aktor (Rosa dan Shalahuddin, 2019)

Diagram Aktivitas mengilustrasikan proses yang terjadi pada sistem baik aktivitas kegiatan utama atau hubungan di antara kegiatan lain (Dennis et al., 2015). Adapun fungsi diagram aktivitas adalah :

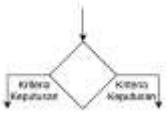
1. Rancangan proses bisnis dalam sebuah sistem
2. Mengurutkan dan mengelompokan tampilan sistem dalam setiap aktivitas

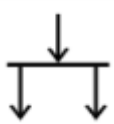
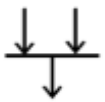
3. Rancangan pengujian sistem, karena setiap aktivitas memerlukan sebuah pengujian

3.2 Notasi Diagram Aktivitas

Dalam membuat Diagram Aktivitas mengilustrasikan alur sistem yang digambarkan dalam sebuah aktivitas dengan aktivitas lain pada suatu proses sistem (Dennis et al., 2015). Berikut adalah Notasi yang dapat digunakan dalam membuat diagram aktivitas:

Tabel 3.1. Notasi Digaram Aktivitas

No	Nama	Notasi	Keterangan
1	Initial Node		Menjelaskan awal aliran sistem dalam suatu proses
2	Final-activity Node		Menjelaskan proses berakhirnya aktivitas dalam sebuah suatu proses
3	Final-flow Node		Menunjukan mengakhiri aliran tetapi bukan mengakhiri aktivitas
4	Action		Perilaku yang sederhana dan tidak dapat diuraikan.
5	Activity		Menunjukkan langkah-langkah aktivitas sistem yang terjadi pada sistem
6	Decision Node		Digunakan jika ada kondisi pilihan yang harus diekseskusi
7	Merge Node		Digunakan untuk menggabungkan berbagai jalur keputusan yang dibuat menggunakan simpul keputusan.

No	Nama	Notasi	Keterangan
8	Fork Node		Digunakan untuk membagi aktivitas menjadi seperangkat aktivitas yang paralel atau bersamaan
9	Join Node		Digunakan untuk menyatukan serangkaian arus aktivitas paralel atau bersamaan akhir dari proses yang berjalan bersamaan
10	Swimlane		Digunakan untuk batasan dalam memecah diagram menjadi baris atau kolom
11	Object Flow		Menunjukkan aliran suatu objek dari satu aktivitas ke aktivitas lain.
12	Control Flow		Menunjukkan urutan eksekusi atau arus dari tiap proses ke proses lain.
13	Object Node		Digunakan untuk mewakili suatu objek yang terhubung ke sekumpulan arus objek.

Sumber : Dennis et al., 2015

3.3 Langkah Pembuatan Diagram Aktivitas

Tahapan dalam membuat diagram aktivitas diantaranya adalah (Nugroho, 2009) dan (Fowler, 2004):

1. Membuat notasi initial node untuk mengawali diagram
2. Membuat notasi action pertama, kedua dan seterusnya sesuai aliran sistem. Dapt menggunakan notasi fork jika ada berbagai aktivitas yang akan diekskusi secara bersamaan. Kemudian aktivitas dapat digabungkan dengan notasi join.
3. Dapat menggunakan Percabangan keputusan untuk memenuhi kondisi tertentu.
4. Aktivitas diakhiri dengan notasi finalnode

5. Jumlah Alur pada diagram aktivitas dengan deskripsi use case haru sama

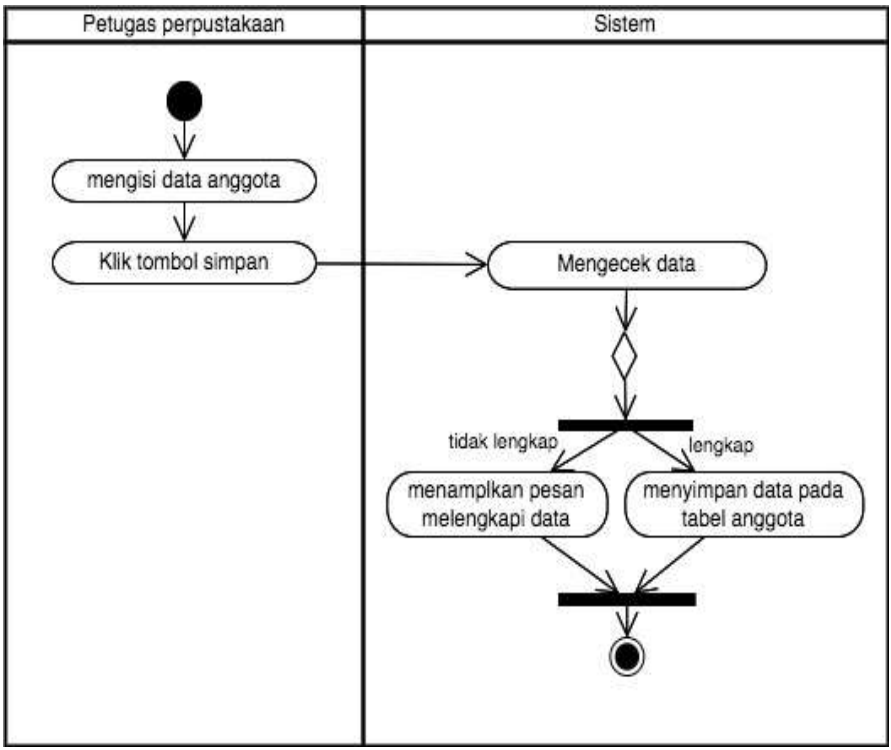
3.4 Studi Kasus : Membuat Diagram Aktivitas

Berikut contoh pembuatan diagram aktivitas pada sistem perpustakaan yaitu pengelolaan data anggota, peminjaman dan pengembalian buku.

Tabel 3.2. Deskripsi Usecase Pengelolaan data anggota

No. Usecase	:	1
Nama Usecase	:	Pengelolaan Data Anggota
Aktor	:	Petugas Perpustakaan
Deskripsi Usecase	:	Usercase ini menjelaskan alur sistem pada saat pengelolaan data anggota
Pre Condition	:	Tampilan Formulir Pengelolaan pendaftaran anggota
Post Condition	:	Data anggota berhasil tersimpan pada tabel anggota
alur berjalanya sistem		
Aksi Aktor		Reaksi Sistem
1. Melakukan pengisian data asesuai dengan formulir		
2. Melakukan Klik tombol simpan		
		3. Mengecek data yang dimasukan oleh aktor dan data sudah sesuai atau tidak
		4. Apabila ada yang belum diisi, maka sistem akan memunculkan pesan ”Data belum lengkap, silahkan perbaiki”
		5. Apabila data yang dimasukan sudah sesuai dan lengkap, data isian anggota tersimpan pada tabel anggota

Berikut Dibawah ini Diagram Aktivitas beradasarkan deskripsi usecase pengelolaan data anggota sebagai berikut :



Sumber : Buku Ajar Ansi, 2018

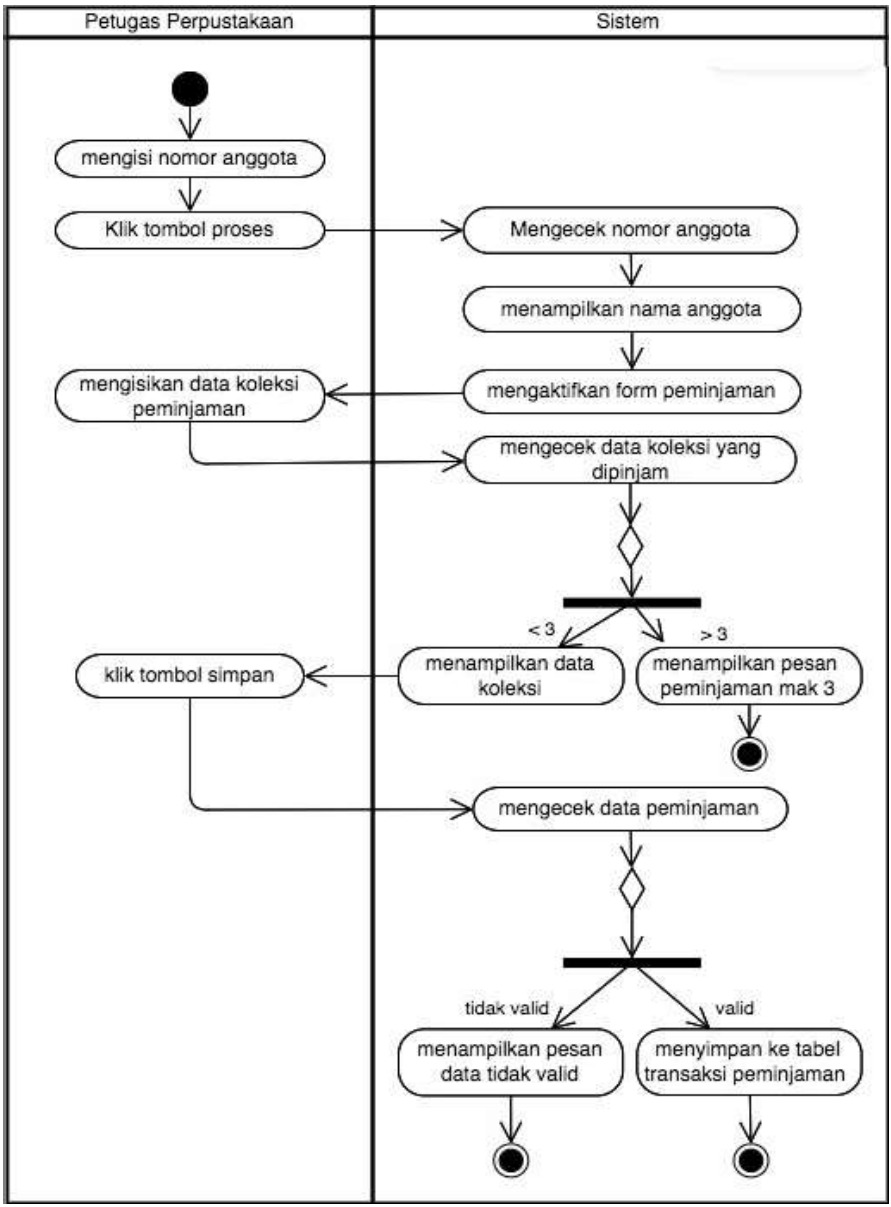
Gambar 3.1. Diagram Aktivitas mengolah data anggota

Tabel 3.3. Deskripsi Usecase Peminjaman buku

No. Usecase	:	2
Nama Usecase	:	Mencatat peminjaman
Aktor	:	Petugas perpustakaan
Deskripsi Usecase	:	Usecase tersebut menjelaskan alur sistem dalam peminjaman buku
Pre Condition	:	Tampilan Formulir peminjaman buku
Post Condition	:	Data peminjaman buku berhasil tersimpan pada tabel peminjaman
Alur berjalannya Usecase		

Aksi Aktor	Reaksi Sistem
1. Mengisi kode anggota	
2. Mengklik tombol Proses	
	3. Mengecek data kode anggota
	4. Apabila kode sesuai maka sistem akan menampilkan nama
	5. Apabila kode berbeda, maka sistem menampilkan ”Data tidak sesuai, silahkan Periksa Kembali”
	6. Apabila kode sesuai, sistem menampilkan form peminjaman
7. Memilih buku	
	8. Melakukan pengecekan data buku yang dipinjam. Apabila > 3 buku, sistem menampilkan ”Maaf pinjaman mak. 3 buku”
	9. Jika jumlah pinjaman sesuai sistem menampilkan data peminjaman
10. Mengklik tombol simpan	
	11. Sistem melakukan pengecekan data peminjaman. Apabila sudah lengkap maka sistem menyimpan data ke tabel peminjaman

Berikut dibawah ini Diagaram Aktivitas beradasarkan deskripsi usecase pencatatan peminjaman buku sebagai berikut:



Sumber : Buku Ajar Ansi, 2018

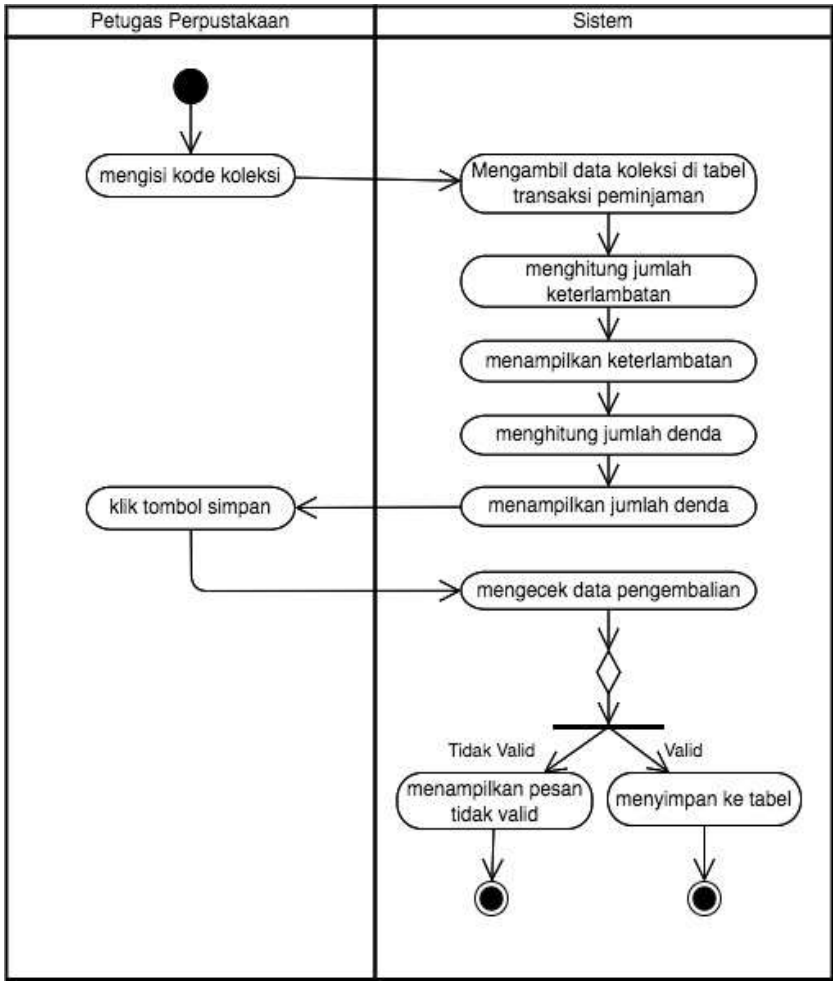
Gambar 3.2. Diagram Aktivitas mencatat peminjaman

Tabel 3.3. Deskripsi Usecase Pengembalian buku

No. Usecase	:	3
Nama Usecase	:	Mencatat pengembalian
Aktor	:	Petugas perpustakaan
Deskripsi Usecase	:	Usecase tersebut menjelaskan alur sistem dalam prose pencatatan pengembalian buku
Pre Condition	:	1. Formulir pengembalian buku 2. Sistem secara otomatis melakukan pengecekan data atas keterlambatan pengembalian
Post Condition	:	Data pengembalian buku berhasil tersimpan pada tabel pengembalian
Alur Berjalanya Usecase		
Aksi Aktor		Reaksi Sistem
1. Mengisi kode peminjaman		
		2. Melakukan pengecekan dan pengambilan data pada tabel peminjaman
		3. Sistem melakukan penghitungan jumlah lama keterlambatan dan akan memampilkan informasinya
		4. Sistem melakukan penghitungan total denda keterlambatan dan akan menampilkan informasi jumlah denda keterlambatan
5. Mengklik tombol Simpan		
		6. Sistem melakukan pengecekan data pengembalian. Apabila sudah sesuai sistem dapat menyimpan data ke tabel pengembalian

	7. Apabila data tidak sesuai, sistem akan menampilkan pesan ”data tidak sesuai”

Berikut dibawah ini Diagaram Aktivitas beradasarkan deskripsi usecase pencatatan Pengembalian buku sebagai berikut :



Sumber : Buku Ajar Ansi, 2018

Gambar 3.3. Activity diagram mencatat pengembalian

DAFTAR PUSTAKA

A. S., Rosa dan Shalahuddin, M. 2019. Rekayasa Perangkat Lunak Terstruktur Dan Berorientasi Objek. Informatika : Bandung.

Buku Ajar, 2018 Analisis dan Perancangan Sistem Informasi. AMIKOM: Yogyakarta

Dennis, A., Wixom, B. H., & Tegarden, D. 2015. Systems Analysis & Design An ObjectOriented Approach with UML (5th Editio). Unitetd States of America : Wiley.

Fowler, M., 2004, UML Distilled A Brief Guide To The Standard Object Modelling Language, Addison-Wesley

Nugroho, Adi. 2009. Rekayasa Perangkat Lunak Menggunakan UML dan Java. Andi : Yogyakarta

Primadewi, Ardhin. Setiya Nugroho. 2022. Sistem Informasi Terintegrasi "Sistem Informasi yang mengintegrasikan data dalam lingkup Enterprise". Magelang : Unimma Press

Winda, Melisa pertiwi., Agung Baitul Hikmah. 2020. Analisa perancangan sistem informasi : penyelesaian studi kasus menggunkan UML. Graha Ilmu : Yogyakarta

BAB 4
PEMODELAN STATE MACHINE DIAGRAM

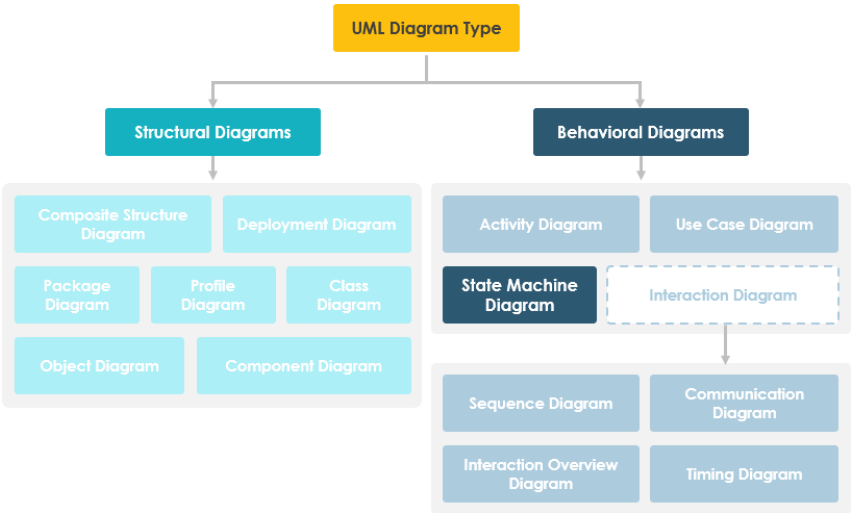
Oleh Wirawan Istiono

4.1 Apa yang dimaksud dengan State Machine Diagram?

Perilaku suatu entitas dipengaruhi tidak hanya oleh masukannya, namun juga oleh keadaan sebelumnya. Lintasan historis suatu entitas dapat direpresentasikan dengan paling akurat oleh state machine diagram terbatas, yang sering dikenal sebagai automata dalam terminologi konvensional. Machine diagram Status UML, sering dikenal sebagai diagram status, State Machine, atau bagan status, menggambarkan berbagai status suatu entitas. State machine diagram dapat mengilustrasikan perilaku dinamis suatu objek saat ia bertransisi antara keadaan yang berbeda sebagai respons terhadap peristiwa yang berbeda. State machine diagram adalah diagram UML yang mewakili perilaku dinamis suatu sistem. (Paradigma, 2023).

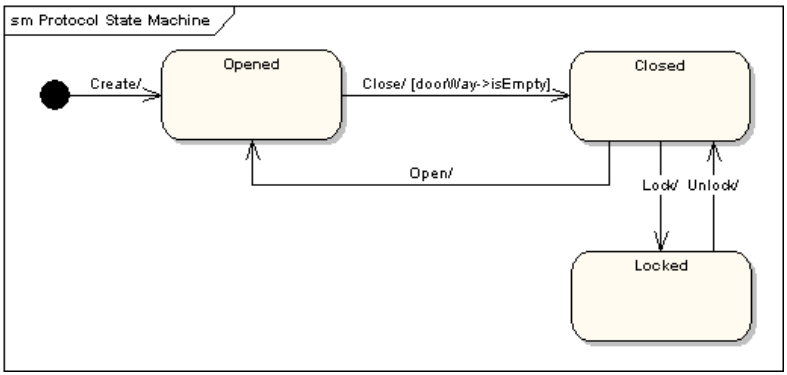
State machine diagram merepresentasikan perilaku objek individu dengan mendefinisikan urutan peristiwa yang dialami objek selama keberadaannya sebagai respons terhadap berbagai peristiwa (Sparxsystems, 2021).

Diagram keadaan terdiri dari keadaan, transisi, peristiwa, dan aktivitas. Diagram keadaan digunakan untuk menggambarkan perspektif dinamis suatu sistem. Mereka memainkan peran penting dalam mewakili perilaku antarmuka, kelas, atau kolaborasi. Diagram status memprioritaskan perilaku berdasarkan peristiwa suatu entitas, yang khususnya bermanfaat dalam mewakili sistem yang responsif (Paradigma, 2023).



Gambar 4.1. *State machine* diagram pada Pohon Diagram Grafik (Sumber: Paradigma, 2023)

Misalnya, state machine diagram berikutnya mengilustrasikan berbagai keadaan yang dilalui sebuah pintu sepanjang keberadaannya.



Gambar 4.2. State machine diagram menggambarkan perkembangan sebuah pintu sepanjang umurnya (Sumber: Sparxsystems, 2021)

Pintu dapat berada dalam tiga keadaan berbeda: "Terbuka", "Tertutup", atau "Terkunci". Sistem mampu bereaksi terhadap kejadian Buka, Tutup, Kunci, dan Buka Kunci. Penting

untuk dicatat bahwa peristiwa tertentu mungkin tidak dapat diterapkan pada tahapan tertentu. Misalnya, jika sebuah pintu terbuka, maka pintu tersebut tidak dapat dikunci sampai pintu tersebut tertutup. Selain itu, penting untuk diperhatikan bahwa transisi suatu state mungkin disertai dengan kondisi penjagaan. Contohnya, saat pintu berada dalam keadaan Terbuka, ia hanya akan merespons kejadian Tutup jika kondisi doorWay->isEmpty terpenuhi. Bagian selanjutnya akan memberikan diskusi komprehensif tentang sintaksis dan standar yang digunakan dalam state machine diagram (Paradigma, 2023).

4.2 Mengapa state machine diagram digunakan?

State machine diagram biasanya digunakan untuk menggambarkan perilaku suatu entitas yang bergantung pada keadaannya saat ini. Suatu objek memperlihatkan respons variabel terhadap peristiwa tertentu yang bergantung pada keadaannya saat ini. State machine diagram biasanya digunakan untuk merepresentasikan perilaku objek, namun juga dapat digunakan untuk elemen apa pun yang berinteraksi dengan entitas lain, seperti aktor, kasus penggunaan, metode, subsistem, dan sistem. Diagram ini biasanya digunakan bersama diagram interaksi, khususnya diagram urutan (Paradigma, 2023).

Sebagai ilustrasi, misalkan Anda memiliki rekening bank sejumlah $100.000. Fungsi penarikan mengurangi Jumlah Penarikan dari saldo, selama saldo yang dihasilkan tidak negatif. Perilaku ini tetap konsisten terlepas dari jumlah penarikan sebelumnya. Dalam keadaan seperti itu, tindakan penarikan tidak berdampak pada generalisasi nilai atribut, sehingga perilaku objek secara keseluruhan tetap tidak berubah.

Namun demikian, jika saldo akun menjadi negatif setelah penarikan, fungsi penarikan akan menunjukkan perilaku yang berbeda. Hal ini terjadi ketika rekening bank berubah dari saldo positif ke saldo negatif. Secara teknis disebut peralihan dari keadaan positif ke keadaan negatif (Paradigma, 2023).

Abstraksi nilai atribut merupakan karakteristik yang melekat pada sistem, bukan prinsip yang berlaku secara

universal. Misalnya, jika bank mengubah peraturan bisnis untuk mengizinkan saldo bank melebihi $2000, status rekening bank akan didefinisikan ulang dengan persyaratan bahwa saldo setelah penarikan tidak boleh kurang dari defisit $2000 (Microtool.de, 2021).

Perlu diketahui bahwa:

1. State machine diagram memberikan representasi komprehensif dari semua peristiwa, keadaan, dan transisi yang terkait dengan satu item.
2. Diagram urutan memberikan representasi komprehensif tentang peristiwa yang terjadi selama interaksi tunggal yang melibatkan banyak item.

4.3 Konsep Dasar State Machine Diagram

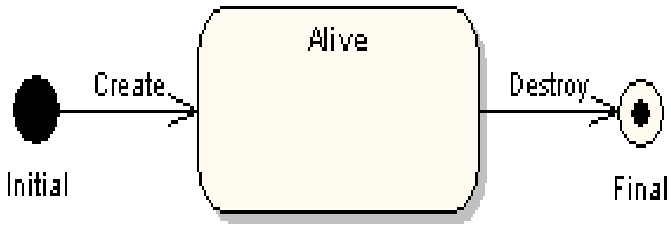
A. Apa itu State?

Menurut definisi dari Rumbaugh "Keadaan merepresentasikan nilai atribut dan koneksi suatu objek dalam bentuk abstrak. Keadaan terdiri dari kumpulan nilai yang dikategorikan berdasarkan properti yang berdampak pada keseluruhan pengoperasian objek." (Paradigma, 2023)

Suatu state direpresentasikan dalam bentuk persegi panjang dengan sudut membulat, termasuk nama state yang tertulis di dalamnya.

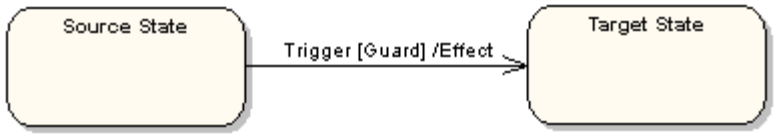
Gambar 4.3. Contoh lambang state
(Sumber: Sparxsystems, 2021)

Initial state diwakili oleh lingkaran hitam pekat dan dapat diidentifikasi dengan sebutan. Keadaan akhir diwakili oleh bentuk lingkaran dengan titik kecil di tengahnya dan juga dapat dikenali dengan nama.



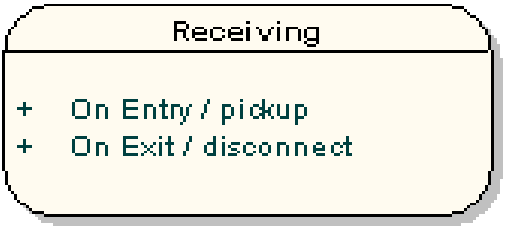
Gambar 4.4. Contoh Initial State dan Final State
(Sumber: Sparxsystems, 2021)

Transisi antar state bagian ditandai dengan garis dengan mata panah. Transisi biasanya terdiri dari tiga komponen: pemicu, penjaga, dan efek.



Gambar 4.5. State dengan Transisi
(Sumber: Sparxsystems, 2021)

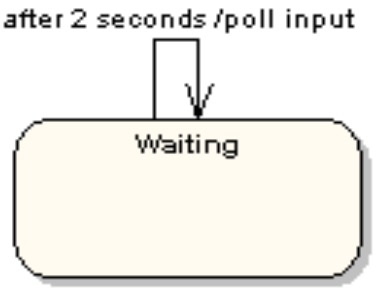
Sebuah "pemicu" adalah faktor yang memulai transisi, seperti sinyal, peristiwa, perubahan situasi, atau berlalunya waktu. "Penjaga" adalah prasyarat yang harus dipenuhi agar pemicu dapat memulai transisi. Sebuah "efek" adalah tindakan yang dipicu langsung pada objek yang memiliki State Machine sebagai konsekuensi dari transisi.
Dalam skenario transisi yang disebutkan di atas, terdapat dampak spesifik yang dikaitkan dengan transisi tersebut. Jika state tujuan menerima banyak transisi, masing-masing dengan efek terkait yang sama, akan lebih menguntungkan untuk mengatribusikan efek tersebut ke state tujuan daripada transisi individual. Tindakan masuk dapat ditentukan bagi state untuk mencapai hal ini. Diagram yang disajikan mengilustrasikan keadaan yang mencakup tindakan masuk dan tindakan keluar (Microtool.de, 2021).



Gambar 4.6. State Actions
(Sumber: Sparxsystems, 2021)

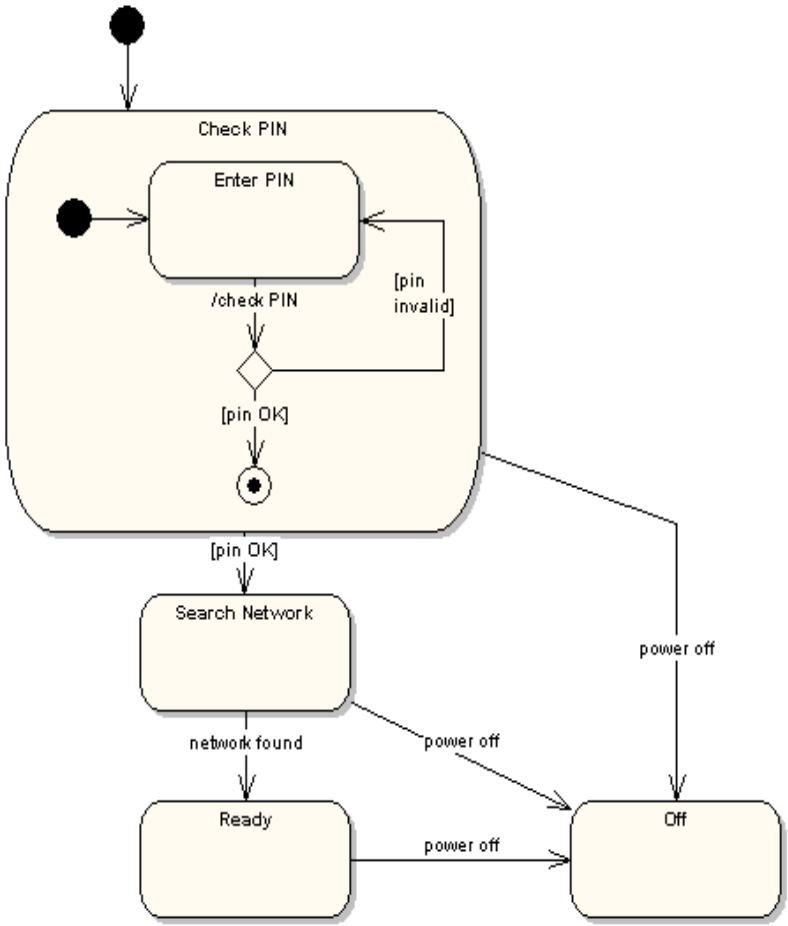
Selain itu, dimungkinkan untuk menetapkan aktivitas yang terjadi sebagai respons terhadap kejadian, atau tindakan yang terjadi secara konsisten. Dimungkinkan untuk menetapkan jumlah tindakan yang tidak terbatas untuk setiap kategori.

Suatu state bagian dapat menunjukkan transisi yang dapat kembali dengan sendirinya, seperti yang digambarkan dalam diagram di bawah. Hal ini sangat bermanfaat ketika terdapat korelasi antara transisi dan hasil.



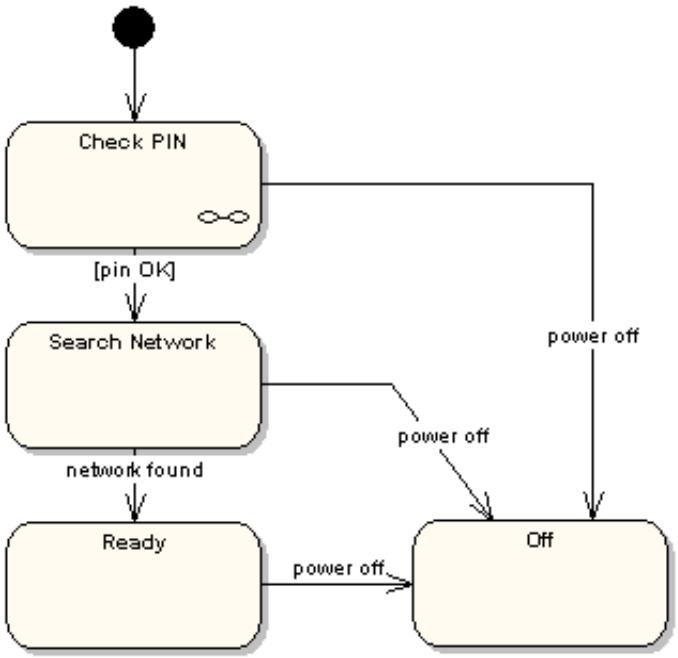
Gambar 4.7. Self-Transitions State
(Sumber: Sparxsystems, 2021)

State machine diagram memiliki kemampuan untuk menggabungkan diagram sub-mesin, seperti yang ditunjukkan dalam contoh berikut.



Gambar 4.8. State Machine Compound
(Sumber: Sparxsystems, 2021)

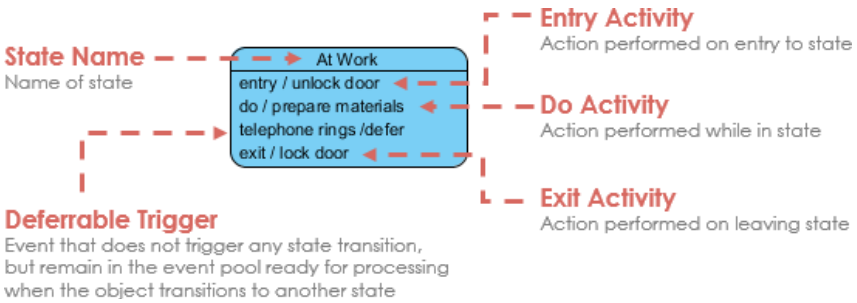
Metode lain untuk menyajikan informasi yang sama adalah sebagai berikut.



Gambar 4.9. State Machine Compound Alternative way
(Sumber: Sparxsystems, 2021)

Notasi pada edisi sebelumnya menandakan bahwa spesifikasi sub-mesin Periksa PIN digambarkan dalam diagram tersendiri.

B. Notasi State



Gambar 4.10. State Notation on State Machine Diagram
(Sumber: Paradigma, 2023)

4.4 Karakteristik Notasi State Machine

Beberapa karakteristik mendefinisikan state secara umum, terlepas dari klasifikasinya (Paradigma, 2023):

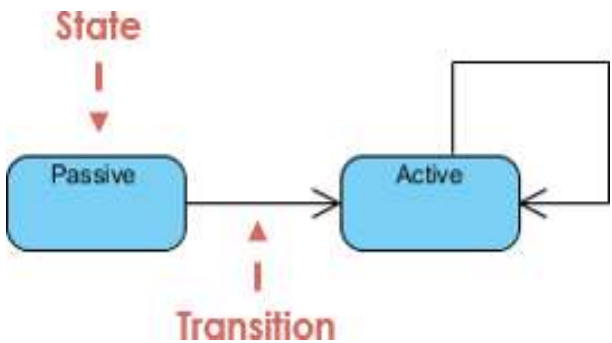
1. State adalah masa transisi.
2. Seringkali, abstraksi nilai atribut suatu entitas yang memenuhi kondisi tertentu dikaitkan dengan suatu keadaan.
3. Keadaan suatu entitas tidak semata-mata ditentukan oleh masukan saat ini, namun juga dipengaruhi oleh rangkaian masukan sebelumnya.

Keadaan adalah suatu kondisi atau keadaan dalam siklus hidup suatu objek dimana suatu batasan terpenuhi, objek melakukan suatu operasi, atau menunggu terjadinya suatu peristiwa (Paradigma, 2023).

Contoh state machine diagram adalah grafik berikut:

1. State bagian sederhana atau state bagian gabungan merupakan state bagian.
2. Transisi antar alam yang menyatukan mereka

Sebagai contoh:



Gambar 4.11. State Transisi Pasif dan Aktif
(Sumber: Paradigma, 2023)

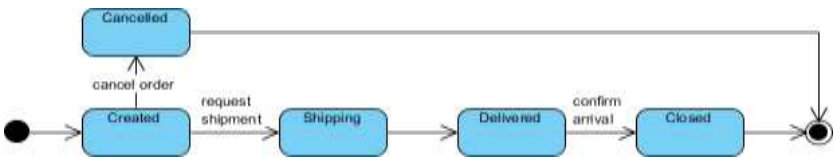
4.5 Atribut State

1. Keadaan adalah keadaan khusus yang dimiliki suatu benda pada waktu tertentu.
2. Objek (atau Sistem) dapat dianggap sebagai transisi antara keadaan yang berbeda.
3. Sebuah elemen model mencapai tahap tertentu dalam siklus hidupnya ketika memenuhi kriteria tertentu, pada titik mana tindakan tertentu dilakukan atau suatu peristiwa ditunggu.

A. Initial and Final States

Keadaan semu awal, yang mewakili titik awal state machine diagram, dilambangkan dengan lingkaran padat. Transisi dari keadaan ini akan mengungkapkan keadaan awal yang nyata (Microtool.de, 2021).

Konfigurasi akhir state machine diagram digambarkan sebagai lingkaran konsentris. State Machine loop terbuka merepresentasikan item yang dapat berhenti secara independen dari sistem, sedangkan machine diagram status loop tertutup tidak memiliki status akhir. Dalam contoh terakhir, objek tetap aktif hingga seluruh sistem berhenti (Paradigma, 2023).



Gambar 4.12. Contoh Diagram Initial dan Final State Machine
(Sumber: Paradigma, 2023)

B. Events

Tanda tangan peristiwa didefinisikan sebagai kombinasi nama peristiwa dan daftar parameter yang dipisahkan dengan koma. Peristiwa terwujud dalam kompartemen transisi internal suatu state bagian atau pada transisi yang menghubungkan state bagian yang berbeda.

Ada empat kemungkinan macam kejadian (Paradigma, 2023):

1. Peristiwa sinyal mengacu pada terjadinya pesan atau sinyal asinkron.
2. Acara panggilan mengacu pada terjadinya panggilan prosedural yang dilakukan ke suatu operasi.
3. Peristiwa waktu adalah kejadian yang terjadi setelah jangka waktu tertentu berlalu.
4. Peristiwa perubahan dipicu ketika kondisi tertentu terpenuhi.

C. Characteristics of Events

1. Menggambarkan kejadian yang mengakibatkan item mengalami perubahan dari satu kondisi ke kondisi lainnya.
2. Peristiwa internal atau eksternal memulai suatu proses yang mengubah status sistem dan beberapa komponen di dalamnya.
3. Peristiwa mengirimkan data, yang selanjutnya diproses oleh tindakan yang dilakukan pada Objek. Objek menjadi sadar atau memahami Kejadian
4. Desain memerlukan analisis peristiwa yang digambarkan dalam state machine diagram dan menentukan bagaimana kejadian tersebut akan difasilitasi oleh entitas sistem.

D. Transition

Garis transisi menggambarkan peralihan atau perubahan dari satu keadaan ke keadaan lainnya. Setiap garis transisi ditandai dengan peristiwa yang memicu transisi tersebut (Paradigma, 2023).

1. Mengingat suatu sistem sebagai kumpulan keadaan dan transisi yang terjadi di antara keadaan-keadaan tersebut sangat bermanfaat untuk menjelaskan perilaku yang rumit.
2. Memahami transisi keadaan merupakan aspek integral dari analisis dan desain sistem.

3. Transisi mengacu pada proses perubahan dari satu keadaan ke keadaan lainnya. Transisi keadaan terjadi dengan cara sebagai berikut (Ali, Maghawry dan Badr, 2021):
   - Suatu elemen berada dalam keadaan sumber.
   - Sebuah insiden terjadi.
   - Sebuah operasi dijalankan.
   - Memulai elemen ke keadaan yang ditentukan
4. Transisi ganda dapat terjadi ketika peristiwa berbeda menyebabkan berakhirnya suatu state, atau ketika kondisi penjagaan diberlakukan pada transisi tersebut.
5. Transisi otomatis mengacu pada perubahan yang terjadi tanpa peristiwa atau tindakan apa pun.

E. Actions

Suatu tindakan mengacu pada perhitungan mandiri dan tidak dapat dibagi yang dapat dieksekusi. Ini mencakup banyak aktivitas seperti menjalankan operasi, membuat atau menghancurkan objek, atau mengirimkan sinyal ke objek. Suatu aktivitas terkait dengan transisi dan terjadi tanpa gangguan, seperti saat masuk atau keluar (Paradigma, 2023).

F. Activity

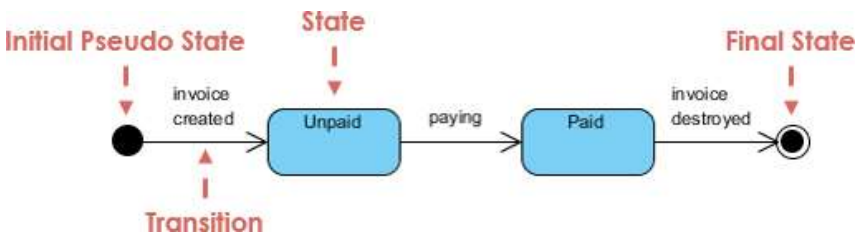
Activity atau aktivitas dikaitkan dengan keadaan, yang mengacu pada komputasi berkelanjutan atau berkelanjutan yang tidak bersifat atomik. Suatu aktivitas dapat mencapai penyelesaiannya atau bertahan selamanya. Suatu Aktivitas berakhir ketika suatu peristiwa memicu transisi dari keadaannya saat ini.

4.6 Atribut Activities dan Actions

1. State mempunyai kemampuan untuk memulai atau menyebabkan tindakan terjadi.
2. State dapat memiliki kompartemen sekunder yang mencakup tindakan atau aktivitas yang dilakukan ketika suatu entitas berada di state tertentu.

3. Suatu tindakan adalah eksekusi yang terpisah dan tidak terputus.
4. Ada lima faktor yang mungkin memulai atau mendorong tindakan: Istilah "On Entry," "Do," "On Event," "On Exit," and "Include" mengacu pada tindakan atau perilaku yang berbeda.
5. Suatu aktivitas mewakili perilaku rumit yang dapat bertahan dalam jangka waktu lama. Hal ini dapat terganggu oleh peristiwa, sehingga kegagalannya terjadi sepenuhnya ketika suatu objek mencapai keadaan tertentu.

4.7 Notasi State machine diagram Sederhana



Gambar 4.13. Notasi untuk State machine diagram Sederhana (Sumber: Paradigma, 2023)

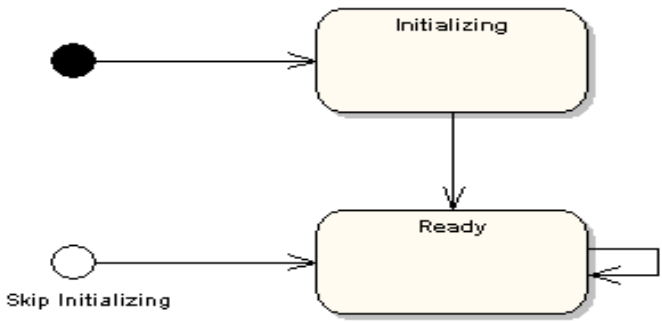
State menentukan kegiatan yang harus dilakukan saat masuk dan keluar. Itu harus benar untuk setiap kejadian masuk atau keluar. Jika tidak, maka Anda diharuskan untuk memanfaatkan tindakan pada setiap busur transisi (Paradigma, 2023).

1. Penyerahan Tindakan yang dilakukan pada saat memasuki keadaan, yang ditunjukkan dengan notasi: Input/operasi
2. Berangkat Tindakan yang dilakukan pada saat keluar dari keadaan, yang ditunjukkan dengan notasi: Pengakhiran/aktivitas

4.8 *Entry* dan *Exit Point* serta Contohnya

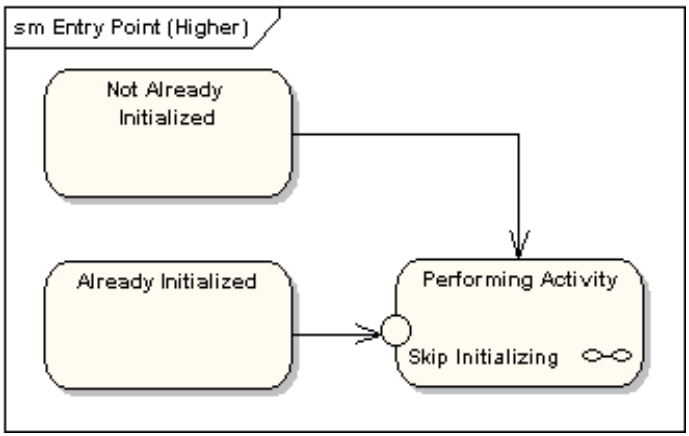
A. Entry Point

Kadang-kadang, Anda mungkin ingin tidak memasukkan sub-mesin dalam keadaan awal yang khas. Misalnya, pada sub-mesin tertentu, biasanya memulai dalam keadaan "Inisialisasi". Namun, jika inisialisasi tidak diperlukan karena alasan apa pun, maka dapat dilakukan dalam keadaan "Ready" dengan beralih ke titik masuk yang ditentukan.



Gambar 4.14. State Machine Titik Masuk
(Sumber: Paradigma, 2023)

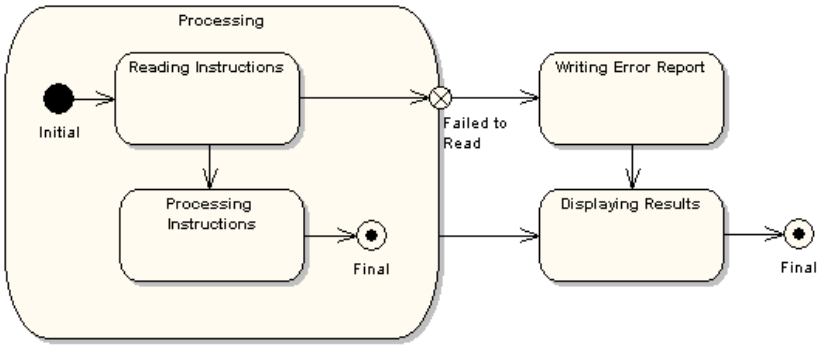
Diagram yang disajikan menggambarkan mesin keadaan pada tingkat yang lebih tinggi (Microtool.de, 2021).



Gambar 4.15. State Machine Entry Point One Level Up
(Sumber: Paradigma, 2023)

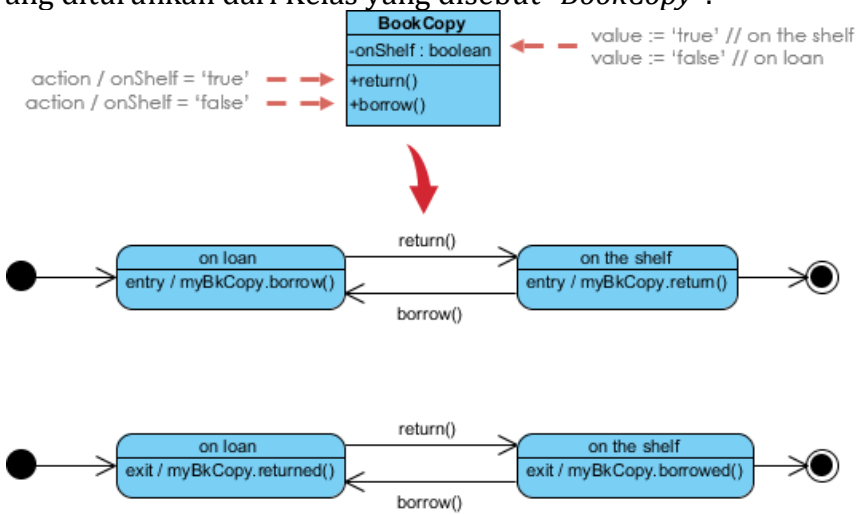
B. Exit Point

Exit Point alternatif yang disebutkan dapat ditetapkan dengan cara yang mirip dengan titik masuk. Grafik mengilustrasikan contoh di mana keadaan yang dieksekusi setelah keadaan pemrosesan utama bergantung pada rute yang dipilih untuk transisi keluar dari keadaan tersebut.



Gambar 4.16. Exit Point State Machine
(Sumber: Paradigma, 2023)

Contoh ini mendemonstrasikan state machine diagram yang diturunkan dari Kelas yang disebut "*BookCopy*".

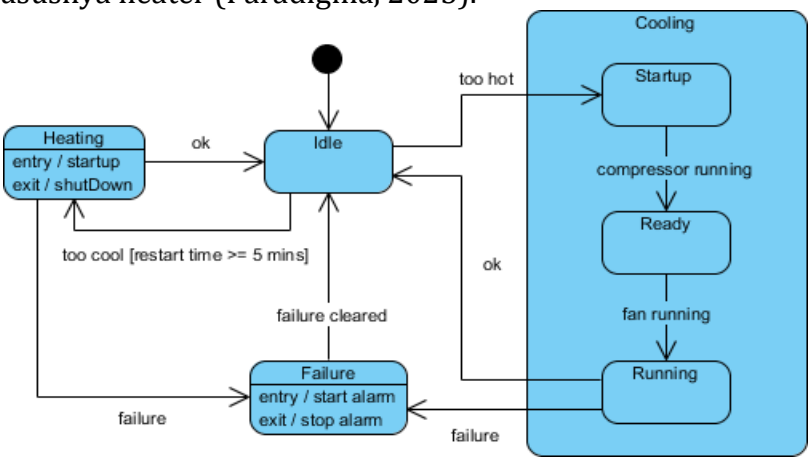


Gambar 4.17. Derivasi state machine diagram dari suatu Kelas
(Sumber: Paradigma, 2023)

1. State machine diagram mengilustrasikan keadaan saat ini dari sebuah objek bernama myBkCopy, yang termasuk dalam kelas BookCopy.
2. Enter action mengacu pada aktivitas apa pun yang ditetapkan untuk dieksekusi setelah keadaan tertentu dimasukkan melalui transisi.
3. Exit action mengacu pada tindakan apa pun yang ditetapkan sebagai tindakan yang terkait dengan tindakan keluar. Tindakan ini dilakukan setelah state berangkat melalui masa transisi.

4.9 Substate bagian

State sederhana ditandai dengan tidak adanya substruktur. State bagian yang berisi substate bagian, juga dikenal sebagai state bagian bersarang, disebut sebagai state bagian gabungan. Substat dapat disarangkan hingga kedalaman berapa pun. State Machine berlapis dapat berisi maksimal satu status awal dan satu status akhir. Substate bagian digunakan untuk menyederhanakan mesin keadaan datar yang rumit dengan menunjukkan bahwa keadaan tertentu secara eksklusif layak dilakukan dalam konteks tertentu, yaitu keadaan terlampir. Gambar 4.18 menampilkan contoh substate, khususnya heater (Paradigma, 2023).



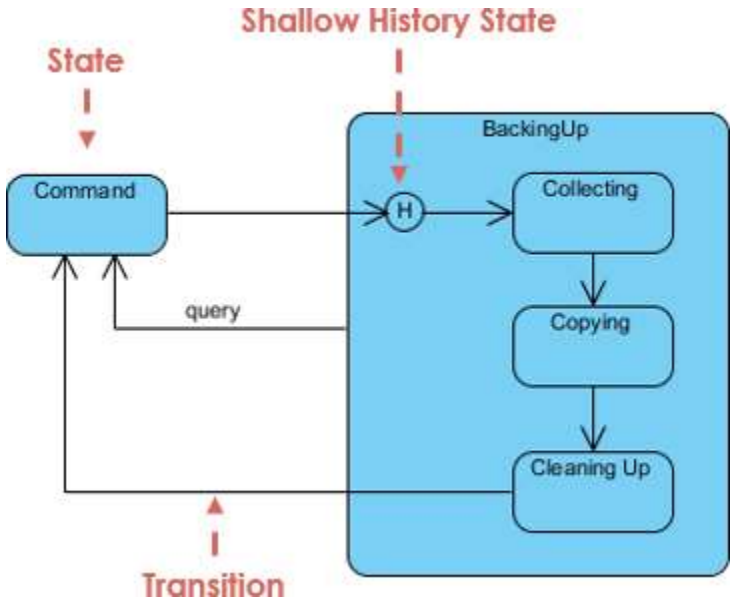
Gambar 4.18. Substate Example - Heater
(Sumber: Paradigma, 2023)

State Machine Diagram sering digunakan untuk mendapatkan skenario pengujian. Di bawah ini adalah kompilasi ide pengujian potensial (Microtool.de, 2021):

1. Status idle dipicu oleh terjadinya peristiwa "Terlalu Panas".
2. Keadaan idle dipicu oleh terjadinya event “Terlalu Keren”.
3. Kondisi Pendinginan/Startup diaktifkan ketika Kompresor diterima. *Running event* Status Pendinginan/Siap menerima Kipas. lomba lari
4. Keadaan pendinginan/berjalan berhasil menerima kejadian OK.
5. Keadaan pendinginan/berjalan diterima. Peristiwa kegagalan
6. Keadaan kegagalan ditandai dengan terjadinya kegagalan. Acara telah terselesaikan.
7. Keadaan pemanasan menerima suatu peristiwa yang menunjukkan bahwa keadaan tersebut berada dalam kondisi memuaskan.
8. Sistem pemanas mengalami kegagalan.

4.10 State Bagian Sejarah

Secara default, ketika transisi memasuki keadaan gabungan, mesin keadaan berlapis akan memulai kembali aksinya dari keadaan semula, kecuali transisi tersebut secara khusus menargetkan suatu subkeadaan. Status riwayat memungkinkan State Machine untuk kembali ke substatus sebelumnya yang aktif sebelum keluar dari status gabungan. Ilustrasi pemanfaatan sejarah state digambarkan pada Gambar 5.19 (Ali, Maghawry dan Badr, 2021).



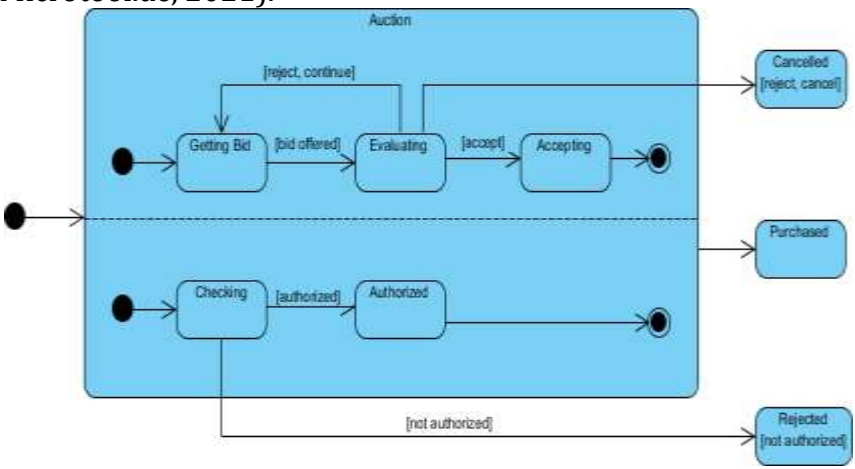
Gambar 4.19. The usage of historical states
(Sumber: Paradigma, 2023)

4.11 Keadaan Bersamaan

Seperti disebutkan sebelumnya, dimungkinkan untuk memiliki status bersarang dalam machine diagram status. State-state yang saling berhubungan dapat dikonsolidasikan menjadi suatu state gabungan yang terpadu. Ketika suatu kegiatan terdiri dari sub-kegiatan yang bersamaan, penting untuk menyatukan keadaan-keadaan di dalam sub-kegiatan yang lain. State machine diagram yang disajikan menggambarkan lelang yang mencakup dua substate bagian secara bersamaan: pemrosesan penawaran dan otorisasi batas pembayaran (Microtool.de, 2021).

Gambar 4.20 menunjukkan contoh Concurrent State Machine Diagram - Proses Lelang. Dalam contoh ini, State Machine awalnya memasuki Lelang dan kemudian dibagi menjadi dua rangkaian awal yang berbeda. Setiap substate dilengkapi dengan exit state yang berfungsi sebagai penanda berakhirnya thread. Keluar dari keadaan gabungan hanya terjadi setelah kedua substate keluar, kecuali jika ada jalan

keluar yang tidak normal seperti pembatalan atau penolakan (Microtool.de, 2021).



Gambar 4.20. Example of a Concurrent State Machine Diagram: Auction Process
(Sumber: Paradigma, 2023)

4.12 Kelebihan dan Kekurangan menggunakan State Diagram

Diagram keadaan dan diagram alur adalah instrumen berharga untuk mengilustrasikan algoritma atau proses, yang masing-masing memiliki kelebihan dan kekurangan yang berbeda. Manfaat menggunakan diagram keadaan adalah (Microtool.de, 2021):

1. Representasi visual berfungsi sebagai sarana untuk menyederhanakan sistem yang rumit dan meningkatkan pemahamannya.
2. Diagram alur seringkali kurang intuitif dan lebih sulit dibaca dibandingkan dengan diagram alur.
3. Diagram ini secara efektif menggambarkan berbagai keadaan di mana suatu sistem mungkin ada dan transisi yang terjadi di antara keadaan-keadaan tersebut.
4. Ini bermanfaat untuk merepresentasikan sistem yang rumit dengan berbagai masukan dan keluaran.

Kerugian dari penggunaan diagram keadaan meliputi (Ali, Maghawry dan Badr, 2021):

1. Mengembangkan dan mempertahankannya dapat menimbulkan tantangan yang lebih besar dibandingkan dengan diagram alur.
2. Meskipun mungkin tidak efektif dalam mengilustrasikan proses linier, tujuan utamanya adalah untuk menggambarkan variasi keadaan.
3. Diagram alur menikmati pengakuan dan penggunaan yang lebih besar dibandingkan dengan diagram alur.

Pemilihan antara diagram keadaan dan diagram alur pada akhirnya bergantung pada kebutuhan sistem yang digambarkan dan preferensi pribadi pengguna. Biasanya, diagram keadaan lebih cocok untuk mengekspresikan sistem yang rumit dengan beberapa masukan dan keluaran, sedangkan diagram alur lebih cocok untuk mengilustrasikan operasi sekuensial.

DAFTAR PUSTAKA

Ali, A., Maghawry, H. A. and Badr, N. 2021. 'Model-Based Test Case Generation Approach for Mobile Applications Load Testing using OCL Enhanced Activity Diagrams', *Proceedings - 2021 IEEE 10th International Conference on Intelligent Computing and Information Systems, ICICIS 2021*. IEEE, (August), pp. 493–499. doi: 10.1109/ICICIS52592.2021.9694216.

Microtool.de. 2021. *Adavantages of State Diagrams*, *microtool.de*. Available at: https://www.microtool.de/en/knowledge-base/what-is-a-state-diagram/ (Accessed: 4 December 2023).

Paradigm, V. 2023. *What is State Machine Diagram?*, *visual-paradigm.com*. Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-state-machine-diagram/ (Accessed: 4 December 2023).

Sparxsystems. 2021. *UML 2 Tutorial - State Machine Diagram*, *sparxsystems.com*. Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-state-machine-diagram/ (Accessed: 4 December 2023).

BAB 5
PEMODELAN SEQUENCE DIAGRAM

Oleh Johni S Pasaribu

5.1 Diagram Interaksi

Dalam membangun sistem informasi menggunakan UML (*Unified Modeling Language*), diagram interaksi berguna untuk memvisualisasikan dan menggambarkan aspek dinamis dari suatu sistem atau aplikasi perangkat lunak. Diagram ini fokus untuk mengilustrasikan bagaimana berbagai objek atau komponen dalam sistem berinteraksi satu sama lain dari waktu ke waktu. Diagram interaksi sangat berguna untuk memodelkan dan memahami aliran pesan, tindakan, dan peristiwa yang terjadi selama eksekusi kasus atau skenario penggunaan.

Diagram interaksi adalah alat penting dalam UML untuk menangkap dan mengkomunikasikan perilaku dinamis suatu sistem. Diagram interaksi membantu pengembang perangkat lunak, perancang, dan pemangku kepentingan memahami bagaimana bagian-bagian berbeda dari suatu sistem berinteraksi dan berkomunikasi selama berbagai skenario, membantu dalam desain, analisis, dan komunikasi perilaku sistem.

Ada dua tipe utama diagram interaksi di UML:

1. Sequence Diagram: Sequence diagram menggambarkan interaksi antara objek atau komponen dalam urutan waktu. Diagram sekuen sering digunakan untuk memodelkan perilaku satu kasus atau skenario penggunaan. Dalam diagram sekuen, aktor maupun objek sebagai garis hidup (garis vertikal, lifeline) dan menunjukkan pesan yang dipertukarkan antara garis hidup ini dari waktu ke waktu. Diagram ini membantu dalam memvisualisasikan urutan kronologis interaksi dan pesan yang dipertukarkan antar objek.
2. Diagram Komunikasi: Diagram komunikasi, juga dikenal sebagai diagram kolaborasi, memberikan perspektif

berbeda tentang interaksi objek. Diagram komunikasi menekankan hubungan struktural antar objek dan bagaimana berkolaborasi untuk mencapai perilaku atau skenario tertentu. Dalam diagram komunikasi, objek digambarkan sebagai node, dan garis penghubung mewakili interaksi atau pesan yang dipertukarkan di antara objek tersebut. Diagram ini berguna untuk memahami hubungan antar objek dalam suatu sistem.

Jadi diagram interaksi digunakan dalam UML untuk menjalin komunikasi antar objek. Itu tidak memanipulasi data yang terkait dengan jalur komunikasi tertentu. Diagram interaksi sebagian besar berfokus pada penyampaian pesan dan bagaimana pesan-pesan ini membentuk satu fungsi sistem. Diagram interaksi dirancang untuk menampilkan bagaimana objek akan mewujudkan kebutuhan tertentu dari suatu sistem. Komponen penting dalam diagram interaksi adalah garis hidup dan pesan.

5.2 Objek, Operasi dan Messages

Dalam model UML, objek adalah elemen yang mewakili instance suatu kelas atau beberapa kelas. Objek adalah turunan kelas pada saat runtime tertentu yang dapat memiliki status dan nilai datanya sendiri. Objek adalah entitas dunia nyata yang perilakunya ditentukan oleh kelas. Objek digunakan untuk mewakili tampilan statis dari sistem berorientasi objek. Kita tidak dapat mendefinisikan suatu objek tanpa kelasnya.

Kelas mewakili hal-hal yang disatukan memiliki perilaku umum. Ini adalah cetak biru dari sistem berorientasi objek. Mari kita perhatikan contoh kelas dan objek yang menarik. Kita semua tahu mobil seperti Ferrari, Volkswagen, Mercedes, dan Porsche, dll. Ini semua adalah perusahaan manufaktur mobil. Di sini, kita dapat menganggap bahwa mobil adalah sebuah kelas dan Ferrari, Mercedes adalah objeknya.

Objek adalah wujud konkret atau *instance* dari kelas, yaitu orang, tempat, atau sesuatu yang informasinya ingin diambil. Misalnya jika ingin membangun sistem reservasi

praktik dokter berbasis web, kelas yang mungkin dapat dibuat adalah dokter, pasien, dan reservasi. Pasien tertentu, seperti Bobby, Adam, dan Siti Hasanah dianggap sebagai objek—yaitu, wujud konkret atau *instance* dari *class* pasien.

Untuk menggambar sequence diagram harus diketahui objek-objek yang terlibat dalam sebuah use case beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu. Banyaknya sequence diagram yang harus digambar adalah sebanyak pendefinisian use case.

Tujuan utama dari pembuatan diagram sekuen adalah untuk mengetahui urutan kejadian yang dapat menghasilkan output yang diinginkan. Selain itu, tujuan dari diagram sekuen ini mirip dengan diagram aktivitas (*activity diagram*), yaitu menggambarkan alur kerja dari sebuah aktivitas, serta dapat menggambarkan aliran data dengan lebih detail, termasuk data atau perilaku yang diterima atau dikirimkan.

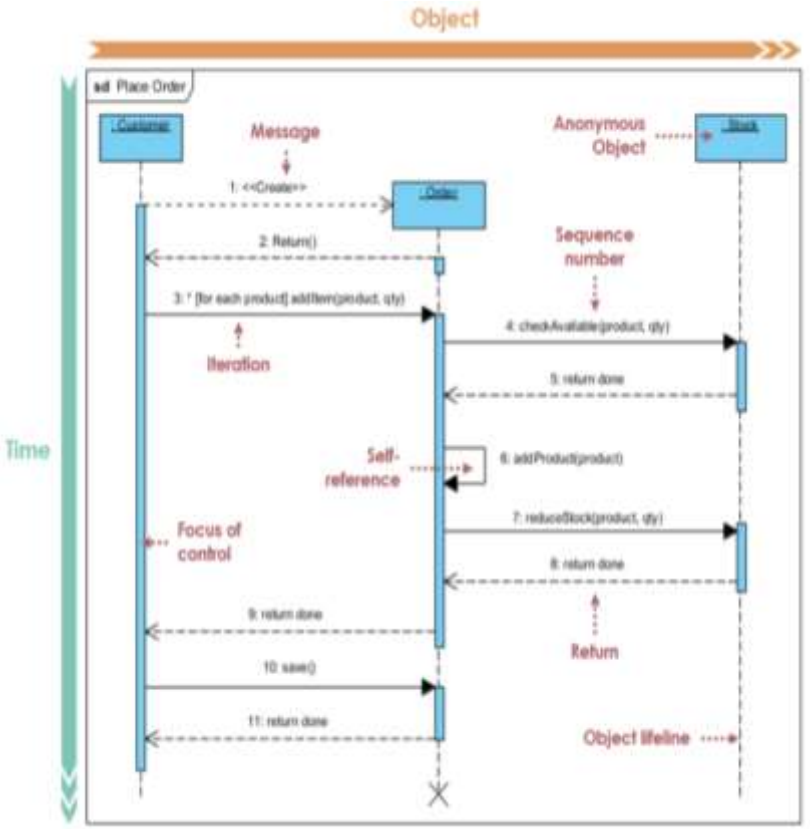
5.3 Sequence Diagram

Sequence Diagram atau Diagram sekuen hanya menggambarkan interaksi antar objek secara berurutan. Tujuan dari diagram urutan dalam UML adalah untuk memvisualisasikan urutan aliran pesan dalam sistem. Diagram sekuen menunjukkan interaksi antara dua garis hidup (*lifeline*) sebagai rangkaian peristiwa yang diurutkan berdasarkan waktu (Ramadhanti, 2023).

Diagram sekuen menunjukkan implementasi suatu skenario dalam sistem. Jalur hidup dalam sistem mengambil bagian selama eksekusi suatu sistem. Dalam diagram sequence, garis hidup diwakili oleh batang vertikal. Alur pesan antara dua objek atau lebih direpresentasikan menggunakan garis titik-titik vertikal yang membentang di bagian bawah.

5.4 Elemen Diagram Sekuen

Misalnya dapat dilihat pada contoh diagram sekuen berikut:



Gambar 5.1. Contoh Diagram Sekuen: Melakukan Pemesanan (Sumber: (Visual Paradigm, 2023))

<u>Contoh Diagram Sekuen: Melakukan Pemesanan</u>

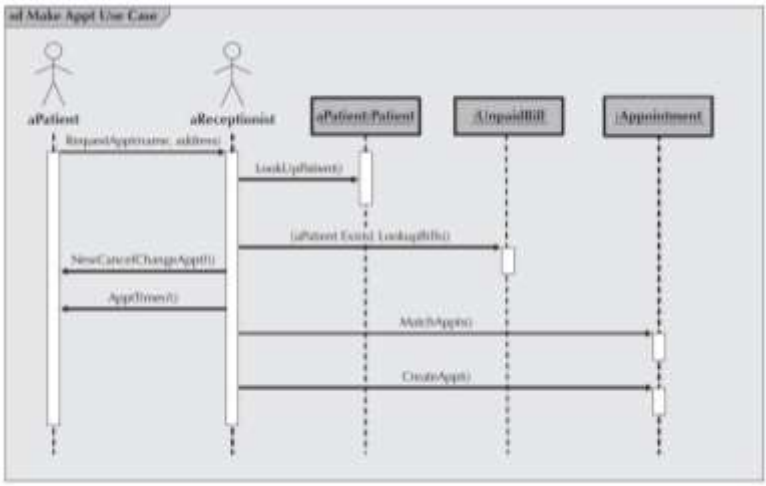
Contoh di atas pada Gambar 5.1 adalah Sequence diagram dengan tiga objek yang berpartisipasi: *Customer* (Pelanggan), *Order* (Pesanan), dan *Stock* (Stok). Langkah-langkah berikut menjelaskan urutan yang terjadi pada Sequence diagram (diagram sekuen) tersebut:

Langkah 1 dan 2: Pelanggan membuat pesanan.

Langkah 3: Pelanggan menambahkan item ke pesanan.

Langkah 4, 5: Setiap item diperiksa ketersediaannya di inventaris.
Langkah 6, 7, 8 : Jika produk tersedia, maka akan ditambahkan ke pesanan.
Langkah 9 *return*
Langkah 10, 11: simpan dan *destroy* (hapus) pesanan

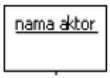
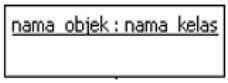
Gambar 5.2 menunjukkan contoh diagram sekuen yang menggambarkan objek dan pesan untuk use case Make Appt, yang menggambarkan proses dimana pasien yang sudah ada (pasien lama) membuat reservasi baru, membatalkan atau menjadwal ulang reservasi untuk sistem reservasi di tempat praktik dokter. Dalam contoh khusus ini, proses Make Appt digambarkan sebagai berikut:

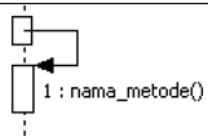


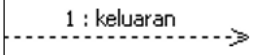
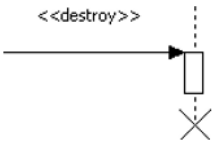
Gambar 5.2. Contoh Diagram Sekuens
(Sumber: (Alan Dennis, 2015))

Perhatikan bahwa aktor pada Gambar 5.2 adalah: aPatient dan aReceptionist sementara objek adalah: aPatient, UnpaidBill, dan Appointment. Untuk masing-masing objek, nama kelas yang menjadi instancenya diberikan setelah nama objek (misalnya, aPatient berarti aPatient adalah turunan dari kelas *Patient*).

Notasi-notasi yang ada pada diagram sekuen:

Simbol	Deskripsi
Aktor  **nama aktor**	orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan
atau  tanpa waktu aktif	orang; biasanya dinyatakan menggunakan kata benda di awal frase nama aktor
Garis hidup / *lifeline*	menyatakan kehidupan suatu objek
Objek 	menyatakan objek yang berinteraksi pesan
Waktu aktif 	menyatakan objek dalam keadaan aktif dan berinteraksi pesan

Pesan tipe create <<create>>	menyatakan suatu objek membuat objek yang lain, arah panah mengarah pada objek yang dibuat
Pesan tipe call 1 : nama_metode()	menyatakan suatu objek memanggil operasi/metode yang ada pada objek lain atau dirinya sendiri
 	arah panah mengarah pada objek yang memiliki operasi/metode, karena ini memanggil operasi/metode maka operasi/metode yang dipanggil harus ada pada diagram kelas sesuai dengan kelas objek yang berinteraksi
Pesan tipe send 1 : masukan	menyatakan bahwa suatu objek mengirimkan data/masukan/ informasi ke objek lainnya, arah panah mengarah pada objek yang dikirimi

Pesan tipe return 	menyatakan bahwa suatu objek yang telah menjalankan suatu operasi atau metode menghasilkan suatukembalian ke objek tertentu, arah panah mengarah pada objek yang menerima kembalian
Pesan tipe destroy 	menyatakan suatu objek mengakhiri hidup objek yang lain, arah panah mengarah pada objek yang diakhiri, sebaiknya jika ada create maka ada destroy

Gambar 5.3. Notasi Diagram Sekuen
(Sumber: (A. S., 2018))

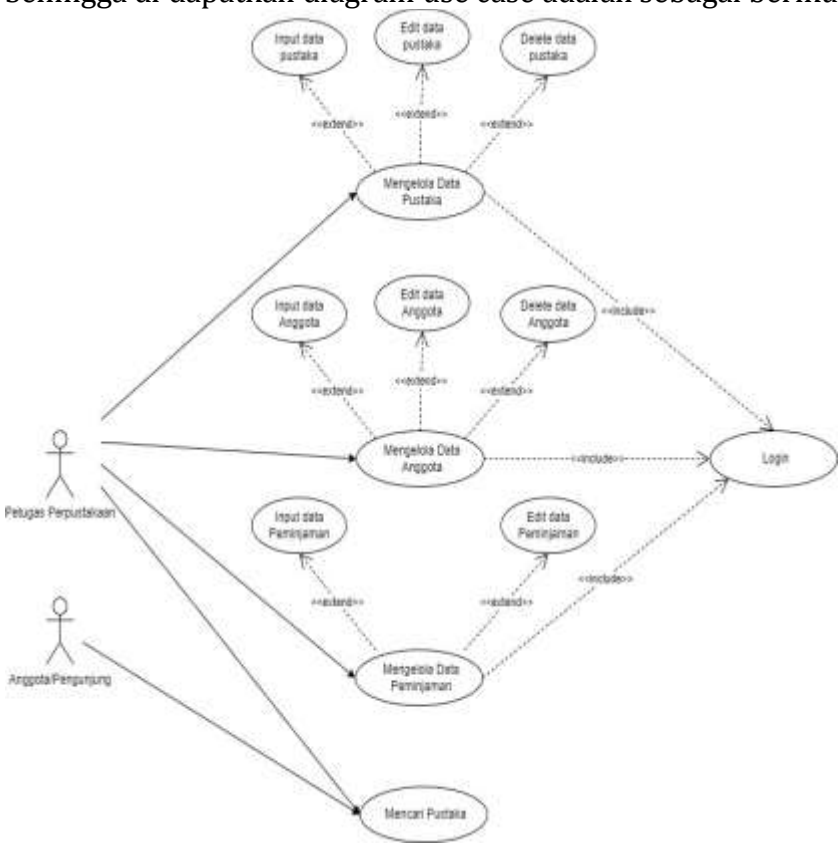
5.5 Studi Kasus Diagram Sekuen

Sistem informasi perpustakaan yang akan dibuat berikut adalah aplikasi berbasis web dimana meliputi pendaftaran pustaka, anggota, dan proses peminjaman pustaka. Petugas perpustakaan melakukan: proses pendaftaran buku (pustaka), anggota, dan peminjaman. Pengunjung maupun anggota dapat melakukan pencarian pustaka namun yang bisa meminjam buku maksimal sebanyak 3 buku, dengan masa peminjaman adalah 7 hari hanyalah anggota perpustakaan. Pengunjung tidak diperbolehkan meminjam pustaka. Tugas petugas perpustakaan yaitu menginput data peminjam dimana akan memeriksa status peminjaman, jika masih dalam masa peminjaman maka anggota tidak dapat meminjam buku.. Fungsi-fungsi dari fitur-fitur sistem informasi perpustakaan adalah sebagai berikut:

1. Mengelola data buku, meliputi:
    a. Masukkan data buku
    b. Ubah data buku
    c. Hapus data buku
2. Mengelola data anggota, meliputi:
    a. Masukkan data anggota
    b. Ubah data anggota
    d. Hapus data anggota

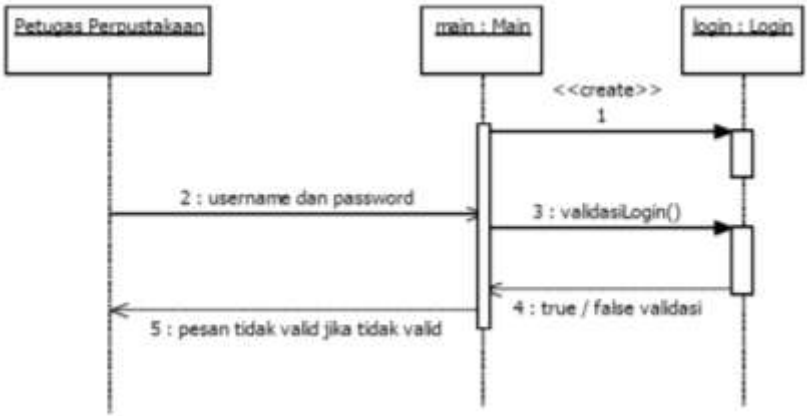
3. Mengelola data peminjaman buku, meliputi:
   a. Masukkan data peminjaman
   b. Ubah data peminjaman (mekanisme pengembalian pustaka)
4. Mencari pustaka

Sehingga di dapatkan diagram use case adalah sebagai berikut

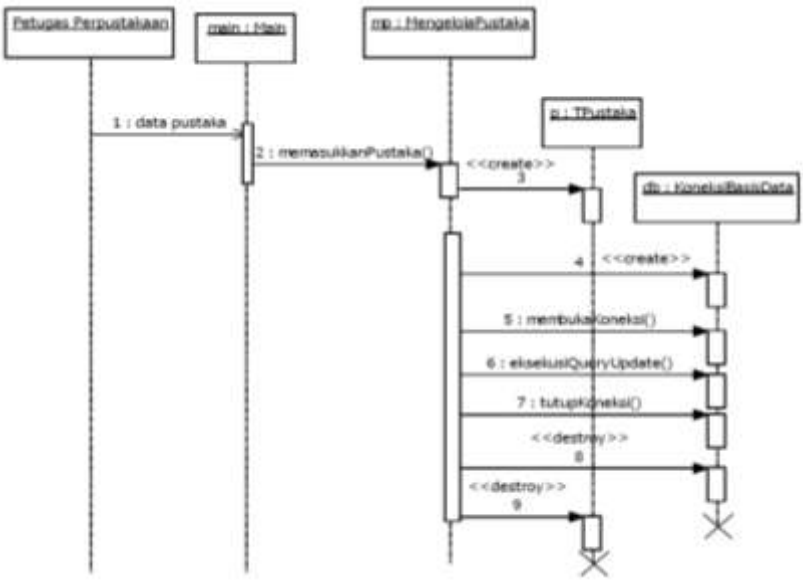


Sequence diagram (diagram sekuen) dari sistem informasi perpustakaan berbasis web tersebut adalah:

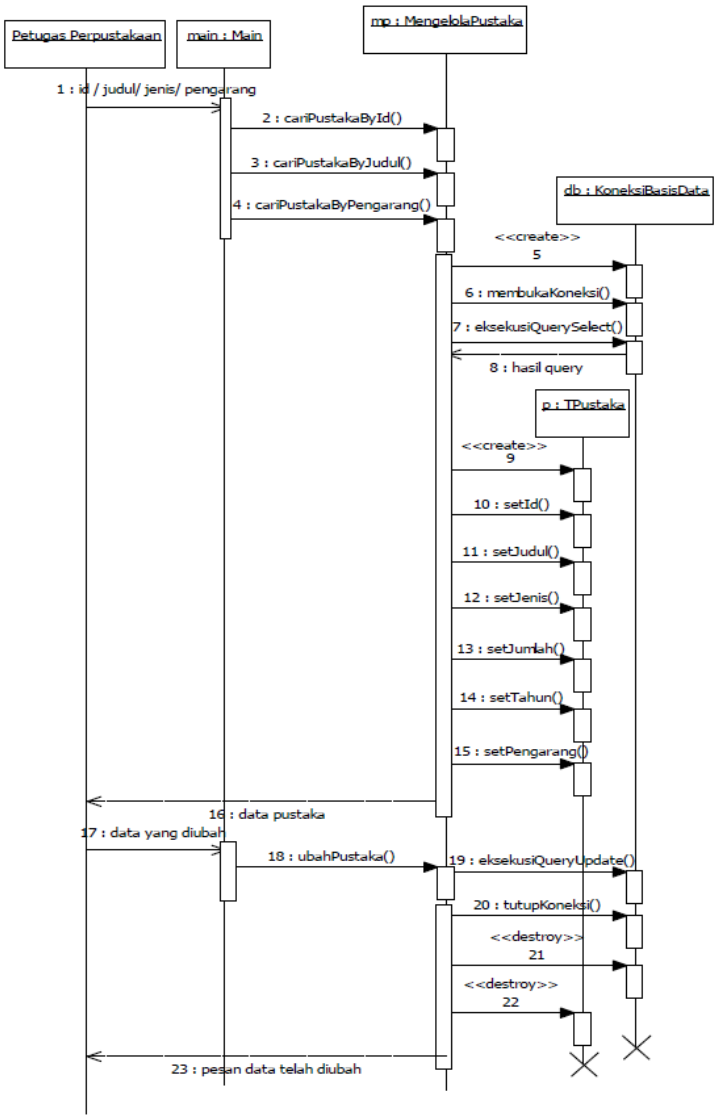
Use case: Login



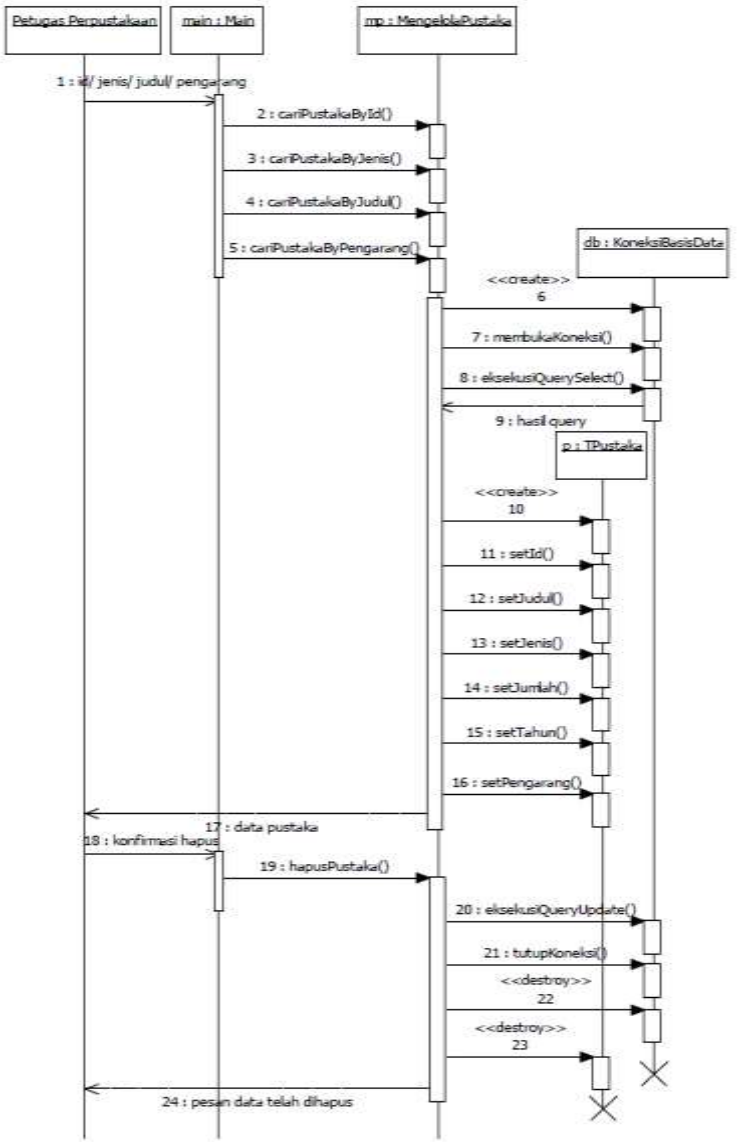
Use case: Memasukkan data pustaka



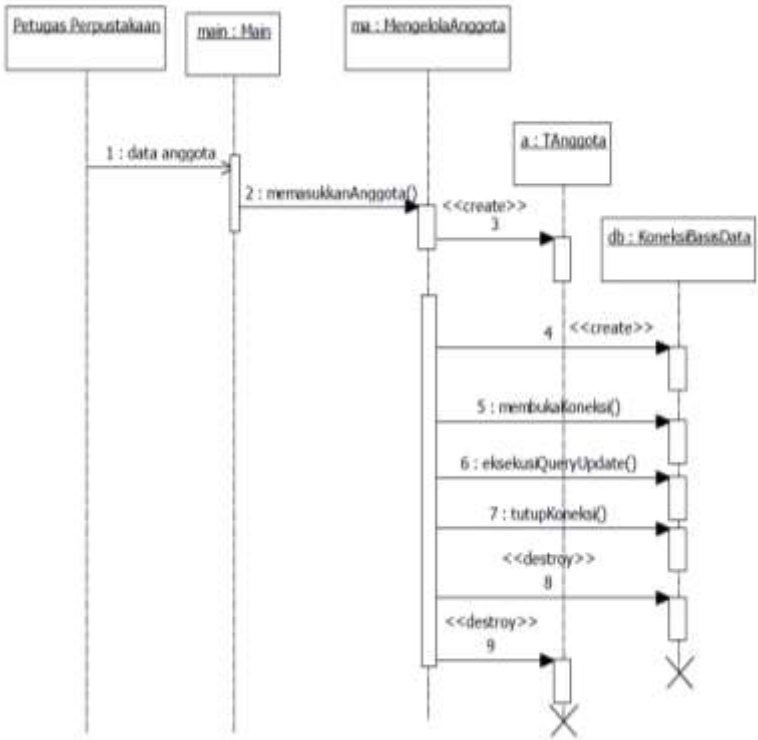
Use case: Mengubah data pustaka



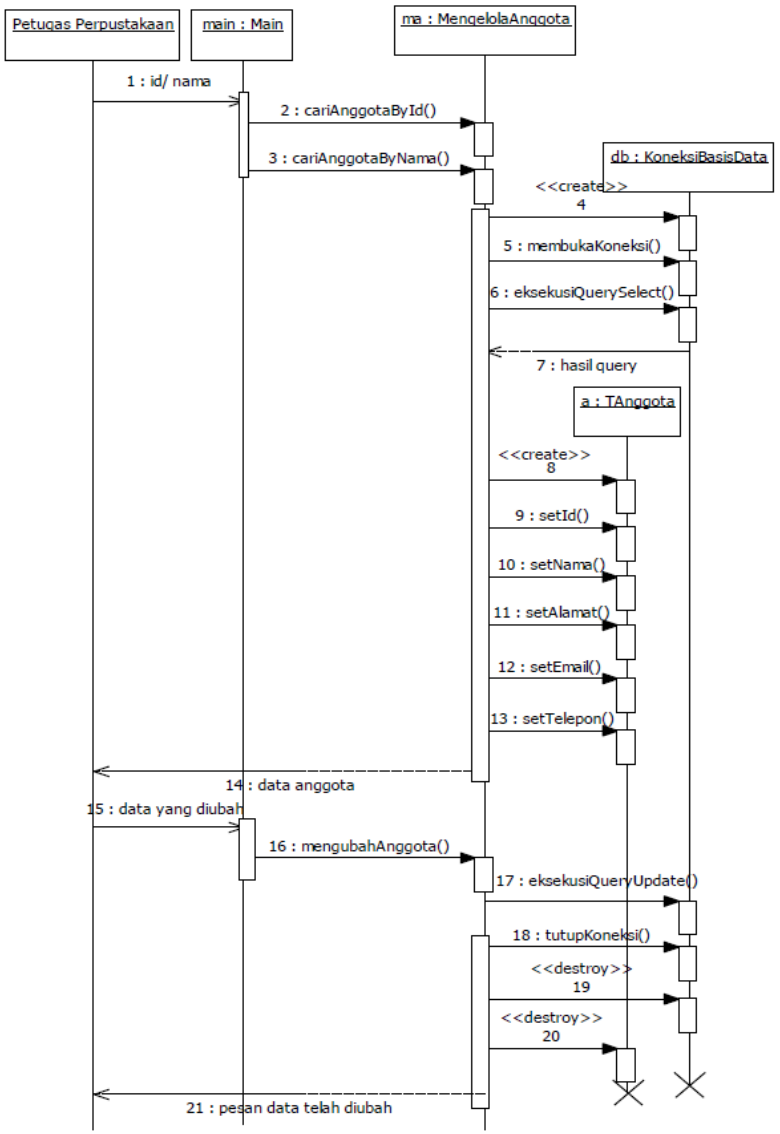
Use case: Menghapus data pustaka



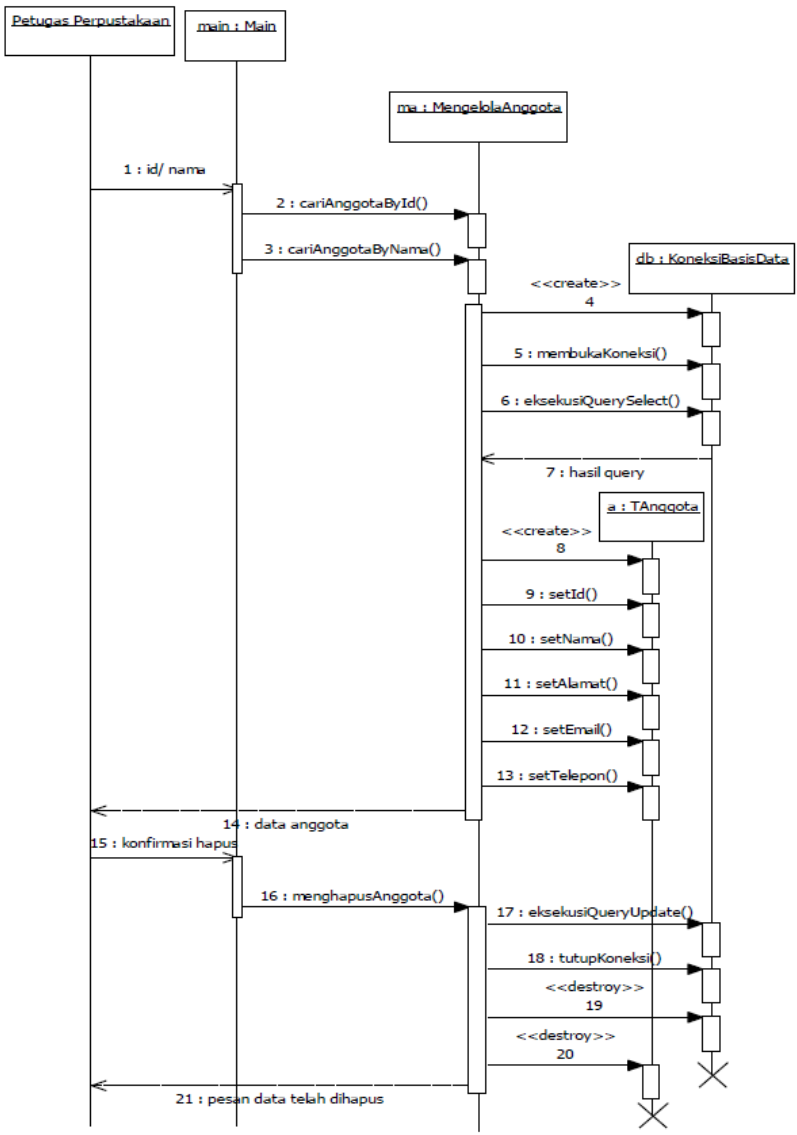
Use case: Memasukkan data anggota



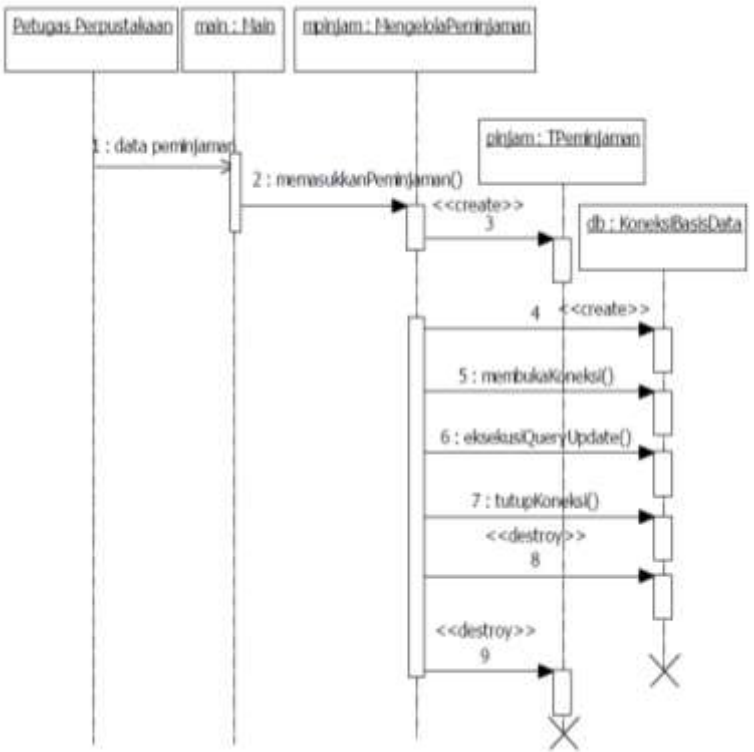
Use case: Mengubah data anggota



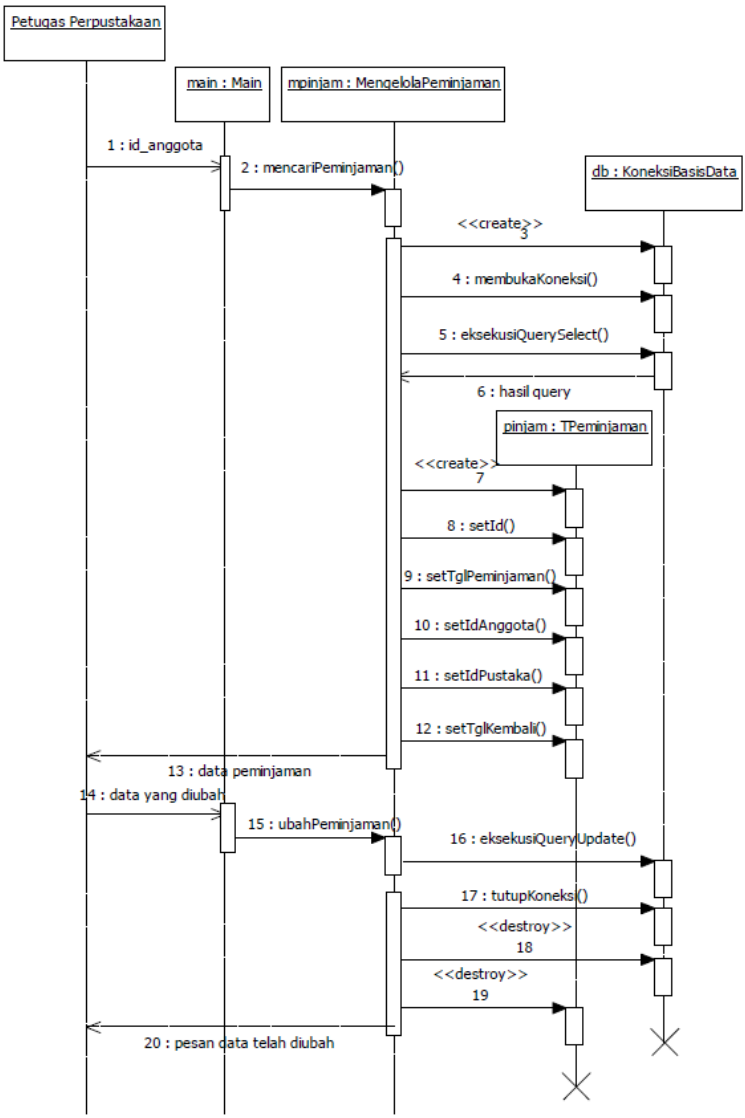
Use case: Menghapus data anggota



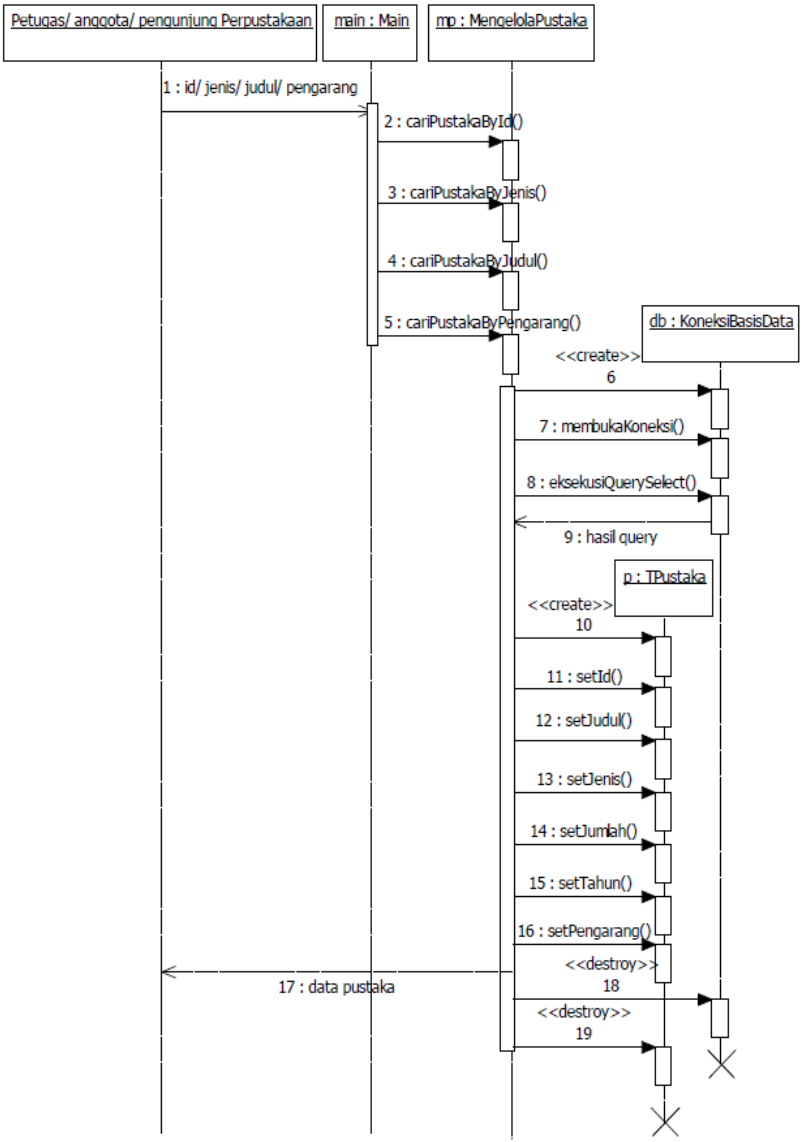
Use case: Memasukkan data peminjaman



Use case: Mengubah data peminjaman



Use case: Mencari pustaka



DAFTAR PUSTAKA

A. S., R. d. 2018. *Rekayasa Perangkat Lunak Terstruktur dan Berorientasi Objek.* Bandung: Informatika.

Alan Dennis, B. H. 2015. *Systems Analysis and Design: An Object-Oriented Approach with UML.* New York: Wiley.

Ramadhanti, A. 2023, Mei 26. *Mengenal Komponen-Komponen & Contoh Sequence Diagram.* Retrieved from Pinhome: https://lifestyle.pinhome.id/blog/contoh-sequence-diagram/

Visual Paradigm. 2023, December 09. *Sequence Diagram.* Retrieved from Visual Paradigm: https://www.visual-paradigm.com/learning/handbooks/software-design-handbook/sequence-diagram.jsp

BAB 6
TEKNIK PENGUMPULAN KEBUTUHAN

Oleh Alexander Waworuntu

6.1 Pendahuluan

Pengumpulan kebutuhan adalah proses penting dalam rekayasa perangkat lunak yang bertujuan untuk mengidentifikasi dan mendokumentasikan kebutuhan fungsional dan non-fungsional dari sistem yang akan dikembangkan (Mohy-ud-din et al., 2017). Proses ini sangat penting karena menjadi dasar untuk semua tahapan desain, pengembangan, dan pengujian dalam siklus hidup pengembangan perangkat lunak (Sandfreni & Surendro, 2016). Kegagalan dalam mengidentifikasi kebutuhan dengan benar dapat menyebabkan kesalahan desain, peningkatan biaya, dan penundaan dalam pengembangan, serta masalah kualitas produk akhir (Lane et al., 2016). Oleh karena itu, pengumpulan kebutuhan yang efektif dan akurat adalah kunci untuk memastikan bahwa produk akhir memenuhi ekspektasi pengguna dan stakeholder, serta berfungsi sesuai dengan tujuan yang ditetapkan.

Dalam siklus hidup pengembangan perangkat lunak, tahap pengumpulan kebutuhan terletak di awal siklus, sebelum perencanaan dan desain sistem dimulai. Proses ini memungkinkan tim pengembangan untuk memahami secara mendalam apa yang diharapkan oleh pengguna dan stakeholder dari sistem yang akan dibangun. Informasi yang diperoleh selama pengumpulan kebutuhan membantu dalam merancang arsitektur sistem, menentukan fitur-fitur kunci, dan merumuskan strategi pengujian. Selain itu, kebutuhan yang terdefinisi dengan baik dan terdokumentasi mempermudah komunikasi antar tim dan membantu dalam mengelola ekspektasi. Dengan demikian, pengumpulan kebutuhan tidak

hanya mempengaruhi kualitas produk akhir, tetapi juga efisiensi proses pengembangan secara keseluruhan, mengurangi risiko, dan membantu dalam manajemen proyek yang efektif.

6.2 Jenis Kebutuhan

Kebutuhan dalam proyek rekayasa perangkat lunak dibedakan menjadi empat kategori utama, yang masing-masing memiliki peranan dan fokus yang berbeda. Pertama, Kebutuhan Fungsional merujuk pada fungsi-fungsi spesifik yang harus dilakukan oleh sistem, seperti proses, detail operasi, dan fitur-fitur yang diharapkan. Kedua, Kebutuhan Non-Fungsional berkaitan dengan aspek-aspek seperti keamanan, performa, keandalan, dan skalabilitas, yang menentukan kualitas sistem secara keseluruhan. Ketiga, Kebutuhan Bisnis menekankan pada kebutuhan organisasi atau perusahaan yang harus dipenuhi oleh sistem, termasuk tujuan-tujuan strategis dan persyaratan pasar. Terakhir, Kebutuhan Pengguna berkonsentrasi pada apa yang diinginkan dan dibutuhkan oleh pengguna akhir, termasuk antarmuka, pengalaman pengguna, dan kebutuhan aksesibilitas. Dengan memahami dan membedakan keempat jenis kebutuhan ini, pengembang dapat menciptakan solusi perangkat lunak yang tidak hanya teknis canggih, tetapi juga sesuai dengan kebutuhan bisnis dan harapan pengguna (Sommerville, 2011).

6.2.1 Kebutuhan Fungsional

Kebutuhan Fungsional dalam konteks rekayasa perangkat lunak merujuk pada spesifikasi fungsi-fungsi yang harus dilakukan oleh sistem atau aplikasi. Ini adalah komponen inti dari setiap proyek pengembangan perangkat lunak, dan mendefinisikan apa yang sistem harus lakukan untuk memenuhi kebutuhan pengguna dan stakeholder (Pressman & Maxim, 2020; Sommerville, 2011). Berikut adalah beberapa aspek penting dari kebutuhan fungsional:

1. Spesifikasi Fungsi Sistem: Kebutuhan fungsional menjelaskan tugas-tugas atau operasi spesifik yang harus dapat dilakukan oleh sistem. Ini termasuk proses seperti input data, pengolahan data, dan output yang dihasilkan.

Misalnya, dalam aplikasi e-commerce, kebutuhan fungsional mungkin termasuk kemampuan untuk menambahkan item ke keranjang belanja, melakukan pembayaran, dan melacak pesanan.

2. Interaksi Pengguna: Kebutuhan ini juga mencakup bagaimana sistem berinteraksi dengan penggunanya, termasuk antarmuka pengguna dan langkah-langkah proses yang harus diikuti. Ini penting untuk memastikan bahwa aplikasi atau sistem mudah digunakan dan intuitif.
3. Integrasi dengan Sistem Lain: Dalam beberapa kasus, kebutuhan fungsional mungkin termasuk bagaimana sistem berinteraksi dan berintegrasi dengan sistem lain. Misalnya, sistem penjualan mungkin perlu berintegrasi dengan sistem akuntansi.
4. Validasi dan Kesalahan: Detail tentang bagaimana sistem harus menangani validasi input dan tindakan saat terjadi kesalahan juga termasuk dalam kebutuhan fungsional.
5. Pemenuhan Aturan Bisnis: Sistem harus dikembangkan dengan mempertimbangkan aturan dan logika bisnis tertentu. Ini termasuk proses pengambilan keputusan yang diotomatisasi, perhitungan, dan pemrosesan bisnis khusus.
6. Keterbatasan dan Batasan: Kebutuhan fungsional juga menetapkan batasan operasional sistem, seperti jenis data yang dapat diolah atau batasan penggunaan sumber daya.

Pengidentifikasian kebutuhan fungsional yang tepat sangat penting karena akan langsung mempengaruhi desain, pengembangan, dan pengujian perangkat lunak. Kebutuhan fungsional yang terdefinisi dengan baik membantu memastikan bahwa produk akhir memenuhi ekspektasi dan kebutuhan pengguna, serta mendukung tujuan bisnis secara keseluruhan.

6.2.2 Kebutuhan Non-Fungsional

Kebutuhan Non-Fungsional dalam rekayasa perangkat lunak mengacu pada kriteria yang digunakan untuk menilai operasi suatu sistem, bukan kegiatan spesifik yang dilakukannya. Berbeda dengan kebutuhan fungsional yang fokus pada apa yang sistem lakukan, kebutuhan non-fungsional

berkaitan dengan bagaimana sistem melakukan fungsi-fungsinya. Kebutuhan non-fungsional sangat penting dalam menentukan kualitas keseluruhan dan kepuasan pengguna terhadap sistem (Pressman & Maxim, 2020; Sommerville, 2011). Beberapa aspek penting dari kebutuhan non-fungsional adalah:

1. Kinerja: Menyangkut seberapa cepat sistem merespons permintaan. Ini termasuk waktu pemuatan halaman, kapasitas pemrosesan, dan waktu respons sistem.
2. Keamanan: Meliputi perlindungan data dan sistem dari akses yang tidak sah, pencurian data, dan kerusakan. Ini termasuk enkripsi, autentikasi, dan kebijakan keamanan.
3. Keandalan dan Ketahanan: Berkaitan dengan stabilitas sistem dan kemampuannya untuk beroperasi tanpa kegagalan dalam periode waktu tertentu. Ini termasuk penanganan kesalahan, pemulihan bencana, dan redundansi data.
4. Skalabilitas: Kemampuan sistem untuk menangani beban kerja yang meningkat tanpa mengorbankan kinerja, baik melalui peningkatan sumber daya hardware atau melalui arsitektur yang dapat diperluas.
5. Kemudahan Pemeliharaan: Berkaitan dengan kemudahan di mana sistem dapat diubah, dikoreksi, ditingkatkan, dan diperiksa. Ini termasuk desain modular, dokumentasi yang baik, dan standar kode yang konsisten.
6. Portabilitas: Menyangkut kemudahan sistem untuk dipindahkan dari satu lingkungan ke lingkungan lain, seperti dari platform operasi satu ke yang lain.
7. Kesesuaian Standar: Kepatuhan terhadap standar dan regulasi industri tertentu, seperti standar aksesibilitas, regulasi privasi data, dan standar interoperabilitas.
8. Pengalaman Pengguna: Menyangkut aspek desain antarmuka pengguna, estetika, dan interaktivitas yang mempengaruhi pengalaman langsung pengguna ketika menggunakan sistem.

Kebutuhan non-fungsional ini sering kali dianggap sama pentingnya dengan kebutuhan fungsional karena mereka secara langsung mempengaruhi kepuasan pengguna, kemudahan

penggunaan, dan daya tahan sistem dalam jangka panjang. Kegagalan dalam memenuhi kebutuhan non-fungsional ini dapat menyebabkan penurunan kepuasan pengguna, peningkatan biaya pemeliharaan, dan bahkan kegagalan sistem secara keseluruhan.

6.2.3 Kebutuhan Bisnis

Kebutuhan Bisnis dalam konteks rekayasa perangkat lunak adalah spesifikasi dan kondisi yang harus dipenuhi oleh sistem atau proyek untuk memenuhi tujuan strategis dan operasional organisasi. Kebutuhan bisnis ini berbeda dari kebutuhan fungsional dan non-fungsional, karena lebih fokus pada tujuan bisnis keseluruhan daripada aspek teknis tertentu dari solusi perangkat lunak (Amjad et al., 2018). Berikut ini adalah beberapa aspek penting dari kebutuhan bisnis:

1. Tujuan Strategis: Ini merujuk pada sasaran jangka panjang yang ingin dicapai organisasi melalui implementasi sistem atau proyek. Misalnya, meningkatkan pangsa pasar, mengoptimalkan efisiensi operasional, atau meningkatkan keterlibatan pelanggan.
2. Pemenuhan Kebutuhan Pasar: Kebutuhan bisnis seringkali mencakup pemahaman tentang kebutuhan pasar yang ditargetkan oleh organisasi. Ini mungkin termasuk mengidentifikasi celah di pasar, memenuhi permintaan pelanggan, atau beradaptasi dengan tren pasar.
3. Proses Bisnis Terkait: Menyangkut bagaimana sistem akan mendukung atau meningkatkan proses bisnis yang ada, termasuk otomatisasi tugas, integrasi dengan sistem lain, dan efisiensi operasional.
4. Pemenuhan Regulasi dan Kepatuhan: Banyak organisasi di sektor tertentu harus mematuhi peraturan hukum dan standar industri. Kebutuhan bisnis harus mencerminkan kebutuhan ini untuk memastikan kepatuhan.
5. Pengembalian Investasi (ROI): Ini berkaitan dengan dampak ekonomi dari implementasi sistem, termasuk biaya, penghematan, dan manfaat finansial jangka panjang. ROI adalah aspek penting yang membantu menentukan kelayakan proyek.

6. Risiko dan Kendala: Kebutuhan bisnis juga harus memperhitungkan risiko potensial dan kendala yang mungkin dihadapi selama dan setelah implementasi, termasuk risiko teknis, operasional, dan keuangan.
7. Perubahan Organisasi: Proyek perangkat lunak seringkali memerlukan perubahan dalam struktur atau budaya organisasi. Ini termasuk pelatihan karyawan, perubahan prosedur kerja, dan manajemen perubahan.
8. Sasaran Jangka Pendek dan Jangka Panjang: Kebutuhan bisnis juga mencakup tujuan jangka pendek seperti peluncuran produk, serta sasaran jangka panjang seperti keberlanjutan dan pertumbuhan bisnis.

Memahami dan mendokumentasikan kebutuhan bisnis dengan jelas adalah langkah penting dalam proses pengembangan perangkat lunak. Ini membantu memastikan bahwa solusi yang dikembangkan tidak hanya secara teknis kompeten tetapi juga sejalan dengan visi dan strategi keseluruhan organisasi. Kegagalan dalam mengidentifikasi atau memahami kebutuhan bisnis dengan benar dapat mengakibatkan proyek yang tidak efektif, yang tidak memberikan nilai tambah nyata atau tidak memenuhi ekspektasi bisnis.

6.2.4 Kebutuhan Pengguna

Kebutuhan Pengguna dalam rekayasa perangkat lunak mengacu pada spesifikasi dan kondisi yang diartikulasikan oleh pengguna akhir, yang menggambarkan apa yang mereka butuhkan dan harapkan dari sistem atau aplikasi. Ini berbeda dari kebutuhan bisnis, fungsional, atau non-fungsional, karena secara langsung berkaitan dengan pengalaman dan interaksi pengguna dengan sistem (Maiden, 2008). Aspek-aspek penting dari kebutuhan pengguna meliputi:

1. Antarmuka Pengguna: Merujuk pada bagaimana pengguna berinteraksi dengan sistem, termasuk desain GUI (*Graphical User Interface*), kemudahan navigasi, dan intuitivitas. Ini juga mencakup aspek seperti layout, warna, font, dan elemen visual lainnya yang mempengaruhi pengalaman pengguna.

2. Fungsionalitas dari Perspektif Pengguna: Kebutuhan ini berfokus pada apa yang pengguna ingin lakukan dengan sistem, seperti menginput data, menerima informasi, atau melakukan transaksi tertentu. Ini berbeda dari kebutuhan fungsional teknis karena lebih berfokus pada perspektif pengguna daripada pada detil teknis.
3. Kemudahan Penggunaan (*Usability*): Termasuk kemudahan mempelajari dan menggunakan sistem, serta efisiensi pengguna dalam melakukan tugas. Ini juga meliputi aspek seperti pencegahan kesalahan dan kenyamanan pengguna saat menggunakan sistem.
4. Aksesibilitas: Kebutuhan pengguna harus memperhitungkan aksesibilitas bagi pengguna dengan berbagai kemampuan, termasuk desain yang mendukung pengguna tunanetra, tuna rungu, atau dengan keterbatasan motorik.
5. Kebutuhan Khusus: Meliputi kebutuhan spesifik yang mungkin diungkapkan oleh kelompok pengguna tertentu, seperti pengaturan kustomisasi, dukungan bahasa, atau fitur-fitur khusus yang memenuhi kebutuhan industri tertentu.
6. Dukungan dan Dokumentasi: Pengguna sering membutuhkan dukungan dalam bentuk tutorial, FAQ, dan dokumentasi yang membantu mereka memahami dan menggunakan sistem secara efektif.
7. Umpan Balik dan Penyesuaian: Kebutuhan pengguna dapat berubah seiring waktu, sehingga sistem harus mampu menyesuaikan diri dengan umpan balik dan kebutuhan pengguna yang berkembang.

Memenuhi kebutuhan pengguna merupakan kunci untuk menciptakan produk yang tidak hanya fungsional tetapi juga diterima dan dihargai oleh mereka yang akan menggunakannya. Pengembangan yang berfokus pada pengguna memastikan bahwa produk akhir bukan hanya mencukupi kebutuhan teknis dan bisnis, tetapi juga menyediakan pengalaman pengguna yang memuaskan dan efektif. Kegagalan dalam memahami dan memenuhi kebutuhan pengguna dapat menyebabkan kurangnya

penerimaan dan kepuasan pengguna, yang pada akhirnya mempengaruhi keberhasilan keseluruhan proyek.

6.3 Teknik Pengumpulan Kebutuhan

Teknik pengumpulan kebutuhan yang berbeda-beda dapat dipilih berdasarkan ciri khas proyek, sumber daya yang tersedia, dan preferensi stakeholder (Pfleeger & Atlee, 2009; Pressman & Maxim, 2020). Dalam praktiknya, teknik-teknik ini bervariasi menurut cara pendekatan, orang yang terlibat, dan alat yang digunakan. Wawancara, baik yang terstruktur maupun tidak terstruktur, memungkinkan pengumpulan data langsung dari pengguna dan stakeholder, memberikan wawasan mendalam mengenai kebutuhan dan ekspektasi mereka. Survei dan Kuesioner efektif untuk mengumpulkan informasi dari sejumlah besar orang dalam waktu yang relatif singkat. Workshop dan Sesi Kelompok melibatkan diskusi interaktif dan kolaboratif, seringkali menghasilkan ide-ide baru dan memperdalam pemahaman akan kebutuhan. Observasi dan Job Shadowing memberikan pandangan real-time tentang bagaimana pengguna berinteraksi dengan sistem saat ini atau melakukan tugas-tugas mereka. Analisis Dokumen membantu mengidentifikasi kebutuhan yang sudah terdokumentasi di dalam materi-materi yang sudah ada. Sesi Brainstorming mendorong pemikiran kreatif dan inovatif dalam mengeksplorasi solusi potensial. Use Cases dan Cerita Pengguna menawarkan narasi terstruktur mengenai bagaimana sistem akan digunakan. Terakhir, Prototyping memberikan gambaran konkrit mengenai bagaimana sistem akan berfungsi, memungkinkan pengguna memberikan umpan balik secara langsung. Setiap teknik ini memiliki kekuatan dan keterbatasan yang unik, dan pemilihan yang tepat bergantung pada konteks proyek, kebutuhan pengguna, dan tujuan organisasi.

6.3.1 Wawancara

Wawancara, baik yang terstruktur maupun tidak terstruktur, adalah teknik penting dalam pengumpulan kebutuhan dalam rekayasa perangkat lunak. Kedua jenis

wawancara ini memiliki karakteristik dan kegunaan yang berbeda (Rezzky et al., 2021):

Wawancara Terstruktur

1. Definisi dan Format: Wawancara terstruktur melibatkan serangkaian pertanyaan yang telah ditentukan dan standarisasi. Pertanyaan-pertanyaan ini dirancang sebelumnya dan disajikan kepada semua responden dengan cara yang sama.
2. Keunggulan:
    a. Konsistensi: Karena pertanyaan standar digunakan, wawancara terstruktur memastikan bahwa informasi yang sama dikumpulkan dari semua responden, memudahkan perbandingan dan analisis data.
    b. Efisiensi: Memiliki format tetap memudahkan proses wawancara dan analisis data yang lebih cepat.
3. Keterbatasan:
    a. Kurang Fleksibilitas: Tidak banyak ruang untuk mengeksplorasi jawaban di luar pertanyaan yang telah ditentukan.
    b. Kemungkinan Kehilangan Nuansa: Beberapa informasi rinci atau kontekstual mungkin terlewat karena pertanyaan yang terlalu spesifik.

Wawancara Tidak Terstruktur

1. Definisi dan Format: Wawancara tidak terstruktur lebih fleksibel dan seringkali bersifat percakapan. Pertanyaan tidak sepenuhnya direncanakan sebelumnya dan dapat berubah berdasarkan arah percakapan.
2. Keunggulan:
    a. Fleksibilitas: Memungkinkan penemuan mendalam tentang pendapat, pengalaman, dan kebutuhan responden.
    b. Menangkap Nuansa: Lebih efektif dalam mengeksplorasi nuansa dan konteks yang mungkin tidak terungkap melalui pertanyaan standar.

3. Keterbatasan:
    a. Waktu dan Sumber Daya: Biasanya membutuhkan lebih banyak waktu dan sumber daya untuk melaksanakan dan menganalisis.
    b. Kurang Konsistensi: Karena setiap wawancara bisa berbeda, mungkin sulit untuk membandingkan hasil antar responden.

Gambar 6.1. Ilustrasi wawancara
(Sumber : http://careerbright.com/)

Dalam praktiknya, seringkali kombinasi kedua jenis wawancara ini digunakan. Wawancara terstruktur dapat memberikan data dasar dan konsisten, sedangkan wawancara tidak terstruktur dapat digunakan untuk mengeksplorasi topik lebih dalam dan memahami konteks serta nuansa yang lebih kompleks. Pemilihan jenis wawancara tergantung pada tujuan pengumpulan kebutuhan, sumber daya yang tersedia, dan tingkat kedalaman informasi yang dibutuhkan.

6.3.2 Survei dan Kuisioner

Survei dan Kuesioner adalah teknik pengumpulan kebutuhan yang penting dalam rekayasa perangkat lunak, terutama ketika informasi harus dikumpulkan dari sejumlah besar orang dalam waktu yang singkat. Kedua metode ini memungkinkan pengumpulan data yang terstruktur dan dapat diandalkan, yang sangat berguna dalam menentukan kebutuhan,

preferensi, dan ekspektasi pengguna (Zachariah & Nonyelum, 2020). Berikut adalah beberapa aspek penting dari survei dan kuesioner:

1. Desain Pertanyaan: Pertanyaan dalam survei dan kuesioner harus dirancang dengan jelas dan spesifik untuk menghindari ambiguitas. Pertanyaan dapat bersifat tertutup (pilihan ganda, skala peringkat) atau terbuka (jawaban bebas), tergantung pada jenis informasi yang dibutuhkan.
2. Sampel Populasi: Survei dan kuesioner memungkinkan untuk mencakup berbagai sampel responden, dari sekelompok kecil pengguna hingga populasi yang lebih luas. Pemilihan sampel yang representatif penting untuk memastikan validitas hasil.
3. Distribusi dan Aksesibilitas: Survei dan kuesioner dapat didistribusikan melalui berbagai cara, seperti email, media sosial, atau platform survei online. Aksesibilitas ini memudahkan partisipasi dari berbagai kelompok pengguna.
4. Analisis Data: Data yang dikumpulkan melalui survei dan kuesioner harus dianalisis secara sistematis untuk mengidentifikasi tren, pola, dan wawasan yang dapat membantu dalam proses pengambilan keputusan.
5. Biaya dan Efisiensi: Metode ini seringkali lebih hemat biaya dan efisien waktu dibandingkan dengan teknik pengumpulan kebutuhan lain, terutama ketika berhadapan dengan sejumlah besar responden.
6. Standarisasi: Survei dan kuesioner memberikan hasil yang dapat dibandingkan dan diukur, karena pertanyaan yang standar diberikan kepada semua responden.
7. Keterbatasan: Meskipun berguna, survei dan kuesioner memiliki keterbatasan, seperti kesulitan dalam menangkap nuansa dan pemahaman mendalam terhadap kebutuhan pengguna. Tanggapan bias juga bisa terjadi tergantung pada cara pertanyaan diajukan.
8. Umpan Balik Anonim: Metode ini memungkinkan pengumpulan umpan balik anonim, yang dapat mendorong kejujuran dan keterbukaan dari responden.

Gambar 6.2. Ilustrasi survey dan kuisioner
(Sumber : https://www.thoughtco.com/)

Survei dan kuesioner adalah alat yang sangat efektif untuk mengukur sikap, opini, dan perilaku pengguna terhadap aspek-aspek tertentu dari sistem atau produk. Ketika dirancang dan diterapkan dengan tepat, mereka dapat memberikan wawasan berharga yang membantu dalam merumuskan kebutuhan yang akurat dan relevan untuk pengembangan perangkat lunak.

6.3.3 Workshop dan Sesi Kelompok

Workshop dan Sesi Kelompok adalah teknik pengumpulan kebutuhan dalam rekayasa perangkat lunak yang melibatkan interaksi langsung dan kolaboratif antara berbagai stakeholder, termasuk pengguna akhir, manajer proyek, analis bisnis, dan pengembang. Tujuan utama dari teknik ini adalah untuk memfasilitasi diskusi terbuka, brainstorming, dan pertukaran ide untuk mengidentifikasi, memprioritaskan, dan merinci kebutuhan sistem (Zachariah & Nonyelum, 2020). Berikut adalah beberapa aspek penting dari workshop dan sesi kelompok:

1. Partisipasi Aktif: Workshop dan sesi kelompok mengundang semua pihak terkait untuk berpartisipasi secara aktif. Hal ini

memungkinkan berbagai perspektif dan kebutuhan untuk didengar dan diintegrasikan ke dalam pengumpulan kebutuhan.

2. Fasilitasi: Sesi-sesi ini biasanya difasilitasi oleh seseorang yang berpengalaman dalam memimpin diskusi dan memastikan bahwa semua suara didengar. Fasilitator juga membantu menjaga fokus kelompok pada tujuan pertemuan dan menghindari pembahasan yang tidak relevan.
3. *Brainstorming* dan *Ideation*: Salah satu kegiatan utama dalam workshop adalah brainstorming, di mana peserta didorong untuk berbagi ide dan solusi kreatif. Ini sering menghasilkan solusi inovatif yang mungkin tidak muncul dalam pengaturan yang lebih formal.
4. Penentuan Prioritas: *Workshop* dan sesi kelompok sering digunakan untuk menentukan prioritas kebutuhan. Ini bisa melibatkan pemungutan suara, diskusi, atau teknik penilaian untuk menentukan fitur mana yang paling penting bagi pengguna.
5. Visualisasi dan Prototyping: Dalam beberapa kasus, workshop dapat mencakup pembuatan prototipe awal atau penggunaan alat visualisasi untuk membantu peserta memahami dan merespon konsep atau desain.
6. Dokumentasi: Semua diskusi, ide, dan keputusan yang dihasilkan selama workshop perlu didokumentasikan secara akurat. Dokumentasi ini nantinya akan menjadi bagian penting dari spesifikasi kebutuhan.
7. Membangun Konsensus: *Workshop* merupakan alat efektif untuk membangun konsensus di antara berbagai stakeholder, terutama dalam proyek dengan kepentingan yang beragam atau kompleks.
8. Evaluasi dan Review: Setelah workshop, sering kali ada sesi evaluasi di mana peserta dapat meninjau hasil dan memberikan umpan balik tambahan.

Workshop dan sesi kelompok sangat berguna dalam proyek yang kompleks, di mana pemahaman mendalam tentang kebutuhan dan harapan pengguna sangat penting. Mereka juga

berguna dalam situasi di mana perlu membangun pemahaman bersama atau konsensus di antara berbagai pihak yang terlibat.

8.3.4 Observasi dan Job Shadowing

Observasi dan *Job Shadowing* merupakan teknik pengumpulan kebutuhan yang krusial dalam rekayasa perangkat lunak, memberikan wawasan mendalam tentang praktek kerja sehari-hari pengguna dan interaksi mereka dengan sistem yang ada. Dalam observasi, seorang analis bisnis atau pengembang perangkat lunak akan mengamati pengguna secara pasif saat mereka menjalankan tugas sehari-hari mereka, memungkinkan pengamat untuk memahami konteks penggunaan sistem yang sebenarnya dan mengidentifikasi tantangan, bottleneck, dan proses kerja yang tidak terdokumentasi. Metode ini sangat efektif dalam mengungkap kebutuhan nyata pengguna serta aspek-aspek penting dari lingkungan kerja yang dapat mempengaruhi desain antarmuka pengguna.

Gambar 6.3. Ilustrasi job shadowing
(Sumber : https://www.uow.edu.my/)

Di sisi lain, *Job Shadowing* melibatkan pengamat yang secara aktif mengikuti pengguna dalam pekerjaan sehari-hari mereka, terkadang bahkan berpartisipasi dalam tugas-tugas

untuk mendapatkan pemahaman yang lebih mendalam. Teknik ini memungkinkan pengamat untuk mendapatkan perspektif langsung tentang tugas-tugas khusus, alur kerja, dan tantangan yang dihadapi pengguna, mengungkap kebutuhan yang mungkin tidak diucapkan atau tidak sepenuhnya dipahami. Job Shadowing sangat bermanfaat dalam proyek-proyek yang membutuhkan pemahaman mendalam tentang proses bisnis, seperti dalam pengembangan sistem enterprise yang kompleks. Melalui kedua metode ini, tim pengembangan dapat memperoleh pemahaman yang lebih komprehensif tentang pengalaman pengguna sehari-hari, memastikan bahwa solusi yang dikembangkan tidak hanya teknis memadai tetapi juga intuitif dan efisien untuk pengguna.

6.3.5 Analisis Dokumen

Analisis Dokumen dalam konteks pengumpulan kebutuhan untuk proyek rekayasa perangkat lunak adalah proses yang sistematis untuk mengkaji dan mengevaluasi dokumen yang ada dalam organisasi atau proyek. Teknik ini sering digunakan untuk memahami konteks bisnis, proses operasional, dan kebutuhan yang sudah terdokumentasikan. Ini merupakan cara yang efektif untuk mendapatkan informasi awal dan mendalam tentang sistem yang sedang dikembangkan atau ditingkatkan (Gokhool & Nagowah, 2022). Berikut ini adalah beberapa aspek penting dari analisis dokumen:

1. Jenis Dokumen: Dokumen yang dianalisis bisa beragam, termasuk laporan bisnis, manual operasional, spesifikasi sistem yang sudah ada, catatan pertemuan, kontrak, standar industri, dan kebijakan perusahaan. Dokumen-dokumen ini dapat memberikan informasi tentang praktik saat ini, persyaratan hukum dan regulasi, serta kegagalan dan keberhasilan masa lalu.
2. Pemahaman Konteks: Analisis dokumen membantu memahami konteks di mana sistem akan beroperasi. Ini mencakup pemahaman tentang proses bisnis, alur kerja, dan lingkungan operasional.
3. Identifikasi Kebutuhan: Melalui dokumen, kebutuhan yang belum terpenuhi atau area yang memerlukan peningkatan

dapat diidentifikasi. Ini termasuk kebutuhan fungsional, non-fungsional, dan kebutuhan pengguna.
4. Pembandingan dengan Standar Industri: Analisis ini juga memungkinkan perbandingan praktik saat ini dengan standar industri atau praktik terbaik, yang dapat membantu dalam menentukan arah dan prioritas pengembangan.
5. Pencarian Ketidaksesuaian: Dokumen sering mengungkapkan ketidaksesuaian atau konflik antara proses yang ada dan kebutuhan atau tujuan yang diungkapkan, yang memberikan peluang untuk perbaikan.
6. Rujukan untuk Pengembangan Lebih Lanjut: Informasi yang dikumpulkan melalui analisis dokumen sering menjadi rujukan penting dalam tahap pengembangan selanjutnya, seperti saat merancang arsitektur sistem atau menulis spesifikasi rinci.

Analisis dokumen adalah teknik pengumpulan kebutuhan yang relatif murah dan efisien, terutama di awal proyek, ketika mengumpulkan informasi dasar tentang domain proyek dan persyaratan awal. Teknik ini juga berperan penting dalam memastikan bahwa pemahaman tentang kebutuhan didasarkan pada informasi yang sudah ada dan terverifikasi, sehingga mengurangi risiko kesalahpahaman atau mengabaikan faktor penting.

6.3.6 Sesi Brainstorming

Sesi *Brainstorming* adalah sebuah teknik pengumpulan kebutuhan dalam rekayasa perangkat lunak yang fokus pada generasi ide dan solusi kreatif melalui diskusi kelompok. Teknik ini sangat berguna dalam tahap awal pengembangan proyek, di mana ada kebutuhan untuk mengumpulkan berbagai perspektif dan solusi potensial (Zachariah & Nonyelum, 2020). Berikut adalah beberapa aspek penting dari sesi brainstorming:

1. Partisipasi Kelompok: Sesi *brainstorming* biasanya melibatkan partisipasi dari berbagai stakeholder, termasuk pengembang, analis bisnis, pengguna akhir, dan manajer proyek. Diversitas partisipan membantu dalam mengumpulkan berbagai sudut pandang dan ide.

2. Lingkungan Terbuka dan Bebas: Salah satu prinsip utama dari brainstorming adalah menciptakan lingkungan di mana semua ide dihargai dan tidak ada kritik atau penilaian yang langsung diberikan. Ini mendorong partisipasi aktif dan generasi ide tanpa batas.
3. Generasi Ide: Tujuan utama dari sesi ini adalah untuk menghasilkan sebanyak mungkin ide dalam waktu yang singkat. Ini dapat termasuk ide-ide baru, pendekatan alternatif untuk masalah yang ada, atau peningkatan fitur saat ini.
4. Fasilitasi: Sesi *brainstorming* biasanya membutuhkan fasilitator yang mengatur alur diskusi, memastikan semua suara didengar, dan menjaga agar sesi tetap fokus pada tujuan.
5. Visualisasi dan Dokumentasi: Seringkali, ide-ide yang dihasilkan selama sesi brainstorming dicatat dan divisualisasikan menggunakan papan tulis, flip charts, atau alat digital. Hal ini membantu dalam mengorganisir dan meninjau ide-ide yang dihasilkan.
6. Evaluasi dan Penyempurnaan: Setelah sesi brainstorming, ide-ide yang dihasilkan biasanya dievaluasi dan diprioritaskan. Proses ini dapat melibatkan diskusi lebih lanjut atau voting untuk menentukan ide-ide mana yang paling layak untuk dijelajahi lebih lanjut.
7. Pengembangan Konsep: Ide-ide terpilih kemudian dapat dikembangkan lebih lanjut menjadi konsep yang lebih matang, yang akan digunakan dalam perencanaan dan pengembangan proyek.

Gambar 6.4. Ilustrasi sesi brainstorming
(Sumber : https://www.carltonfields.com/)

Sesi *brainstorming* adalah alat yang sangat efektif dalam mengumpulkan kebutuhan karena memungkinkan tim untuk melampaui batas-batas pemikiran konvensional, mengeksplorasi solusi inovatif, dan mendorong keterlibatan serta kolaborasi di antara semua stakeholder proyek. Teknik ini membantu dalam mengidentifikasi kebutuhan dan kemungkinan yang mungkin tidak muncul dalam metode pengumpulan kebutuhan yang lebih tradisional.

6.3.7 Use Cases dan User Stories

Use Cases dan *User Stories* merupakan dua pendekatan dalam pengumpulan kebutuhan dalam rekayasa perangkat lunak, masing-masing dengan ciri khasnya. *Use Case* adalah deskripsi terperinci tentang interaksi antara sistem dan pengguna atau sistem lain untuk mencapai tujuan tertentu, termasuk langkah-langkah yang dilakukan oleh pengguna dan sistem. Ini menyediakan pandangan rinci tentang proses bisnis dan mengidentifikasi kebutuhan fungsional sistem. *Use Case* biasanya mencakup aktor, prasyarat, alur kerja normal, dan alur kerja alternatif, dan sering digunakan dalam metodologi pengembangan tradisional seperti *Waterfall*.

Di sisi lain, *User Story* merupakan deskripsi singkat dan informal yang diungkapkan dalam bahasa alami tentang fitur sistem dari perspektif pengguna akhir. Formatnya sederhana,

sering diikuti model "Sebagai [jenis pengguna], saya ingin [tujuan] agar [manfaat]", yang memfasilitasi diskusi tentang kebutuhan dan solusi. *User Stories* bertujuan untuk menjaga fokus pada kebutuhan pengguna dan sering digunakan dalam metodologi Agile, seperti Scrum, karena fleksibilitas dan adaptivitasnya (Dwitama & Rusli, 2020).

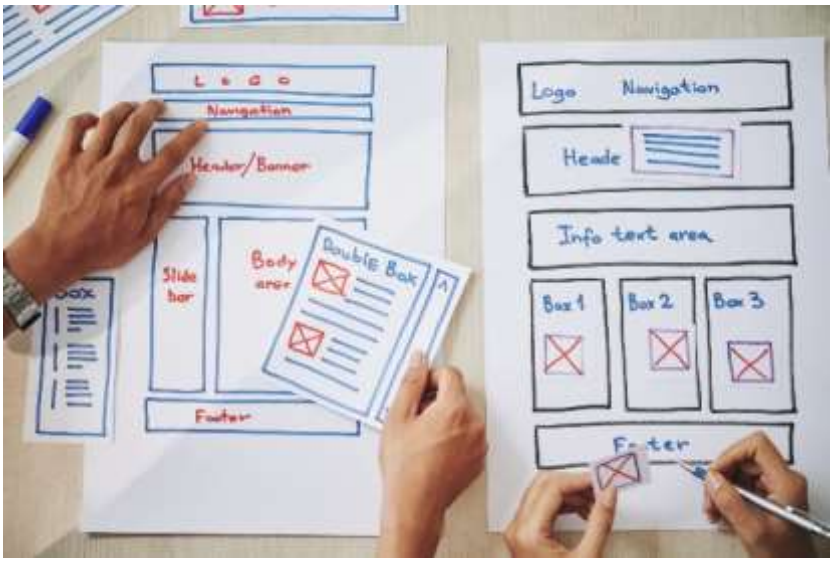
Kombinasi penggunaan *Use Cases* dan *User Stories* dalam proyek rekayasa perangkat lunak dapat memberikan pandangan yang komprehensif dan mendalam tentang kebutuhan dan harapan pengguna, dengan *Use Cases* memberikan kerangka kerja yang lebih rinci dan formal, sementara *User Stories* membawa perspektif pengguna yang lebih kuat dan fokus pada nilai.

6.3.8 Prototyping

Prototyping adalah sebuah pendekatan dalam rekayasa perangkat lunak di mana versi awal dari sebuah sistem, aplikasi, atau produk dibuat untuk menunjukkan ide, fitur, atau fungsionalitas. Proses ini sangat penting dalam pengembangan perangkat lunak karena memungkinkan pengembang dan stakeholder untuk memahami, mengevaluasi, dan mengkomunikasikan kebutuhan dan solusi sebelum pengembangan penuh dilakukan (Deininger et al., 2019). Berikut adalah beberapa aspek penting dari *prototyping*:

1. Model Awal: *Prototype* biasanya adalah versi awal dari sistem yang sedang dikembangkan. Ini bisa dalam bentuk mock-up sederhana atau kerangka kerja awal yang menunjukkan bagaimana sistem akan terlihat dan bekerja.
2. Validasi Konsep: *Prototyping* membantu dalam validasi ide dan konsep sebelum komitmen penuh terhadap pengembangan. Ini memungkinkan pengguna dan stakeholder untuk memberikan umpan balik awal mengenai fitur, desain, dan fungsionalitas sistem.
3. Iteratif dan Evolutif: Proses ini bersifat iteratif. *Prototype* awal dikembangkan, diuji, dan disempurnakan berdasarkan umpan balik, mengarah pada pembuatan prototype yang lebih canggih. Proses ini berulang sampai model yang memuaskan tercapai.

4. Keterlibatan Pengguna: *Prototyping* melibatkan pengguna akhir secara aktif dalam proses desain dan pengembangan. Ini memastikan bahwa produk akhir akan sesuai dengan kebutuhan dan ekspektasi pengguna.
5. Pengurangan Risiko: Dengan mengidentifikasi masalah desain dan persyaratan pada tahap awal, prototyping membantu mengurangi risiko pengembangan dan potensi biaya revisi di masa depan.
6. Pemilihan Teknologi: *Prototyping* juga berguna untuk mengevaluasi berbagai pilihan teknologi dan alat pengembangan sebelum memilih solusi yang paling tepat.
7. Jenis-Jenis *Prototype*: Ada berbagai jenis *prototype*, dari '*low-fidelity*', seperti sketsa atau *mock-up* statis, sampai '*high-fidelity*', yang mirip dengan produk akhir baik dalam tampilan maupun fungsionalitas.
8. Pengujian *Usability*: *Prototype* yang lebih maju dapat digunakan untuk pengujian usability, di mana pengguna diberi tugas untuk menyelesaikan menggunakan prototype untuk menilai kegunaan dan intuitivitas desain.



Gambar 6.5. Ilustrasi prototyping
(Sumber : https://xd.adobe.com/)

Dengan menawarkan tampilan awal tentang bagaimana produk akhir mungkin terlihat dan berfungsi, prototyping merupakan alat yang sangat berharga dalam proses pengembangan perangkat lunak. Ini memungkinkan tim pengembangan untuk membuat keputusan yang lebih tepat tentang desain dan implementasi, sambil memastikan bahwa produk akhir memenuhi kebutuhan dan harapan pengguna.

6.4 Pemilihan Teknik yang Tepat

Dalam proses pengembangan perangkat lunak, pemilihan teknik pengumpulan kebutuhan yang tepat merupakan aspek krusial yang menentukan kesuksesan proyek. Teknik yang dipilih harus efektif dalam menggali kebutuhan yang akurat dan komprehensif, seraya mempertimbangkan faktor-faktor seperti sumber daya, kompleksitas proyek, dan karakteristik stakeholder. Pendekatan ini harus menyesuaikan dengan kondisi unik dan dinamika setiap proyek untuk hasil yang optimal.

Pilihan teknik pengumpulan kebutuhan dipengaruhi oleh berbagai faktor. Kompleksitas dan skala proyek memainkan peran penting; proyek yang lebih besar dan lebih kompleks mungkin memerlukan pendekatan yang lebih mendalam seperti workshop atau prototyping. Sementara itu, karakteristik stakeholder, termasuk ketersediaan dan keahlian mereka, juga menentukan teknik yang paling cocok. Selain itu, batasan sumber daya dan anggaran, serta timeline proyek, juga harus dipertimbangkan dalam memilih teknik yang paling efisien dan efektif.

Teknik pengumpulan kebutuhan juga harus disesuaikan dengan tipe dan ukuran proyek. Proyek kecil mungkin memerlukan pendekatan yang lebih sederhana seperti wawancara atau survei, sementara proyek yang lebih besar mungkin membutuhkan kombinasi dari berbagai teknik. Penting juga untuk menyesuaikan teknik dengan tujuan proyek; misalnya, proyek yang berfokus pada pengalaman pengguna akan sangat diuntungkan dengan prototyping dan pengujian usability.

Penggabungan beberapa teknik pengumpulan kebutuhan seringkali memberikan hasil yang paling komprehensif. Teknik kualitatif seperti wawancara dapat digabungkan dengan teknik kuantitatif seperti survei untuk memberikan pandangan yang luas dan mendalam. Workshop dan brainstorming dapat mengungkapkan ide-ide dan kebutuhan yang mungkin tidak muncul melalui teknik lain, sementara prototyping, diikuti dengan sesi umpan balik, dapat memvalidasi dan memperhalus kebutuhan yang telah dikumpulkan.

Melalui pemahaman yang mendalam tentang setiap teknik dan adaptasi yang cermat terhadap kondisi proyek, tim pengembang dapat memastikan bahwa kebutuhan yang dikumpulkan adalah representasi yang akurat dan mendalam dari apa yang dibutuhkan sistem untuk mencapai tujuan proyek. Pendekatan yang disesuaikan dan beragam ini membantu dalam mengurangi risiko dan meningkatkan probabilitas keberhasilan proyek.

6.5 Tantangan dalam Pengumpulan Kebutuhan

Pengumpulan kebutuhan sering dihadapkan dengan berbagai tantangan yang dapat mempengaruhi efektivitas proses dan kualitas produk akhir. Salah satu tantangan utama dalam pengumpulan kebutuhan adalah menghadapi kebutuhan yang tidak jelas atau tidak lengkap. Hal ini sering terjadi ketika stakeholder tidak yakin dengan apa yang mereka inginkan, atau tidak dapat mengartikulasikan kebutuhan mereka dengan jelas. Untuk mengatasi hal ini, penting untuk melakukan komunikasi yang efektif, menggunakan teknik seperti wawancara mendalam, workshop, dan brainstorming untuk membantu mengklarifikasi dan mendetailkan kebutuhan. Penerapan prototyping juga bisa sangat membantu, dimana stakeholder dapat melihat dan memberi umpan balik terhadap contoh konkret dari apa yang akan dibangun.

Perubahan kebutuhan adalah fenomena umum dalam banyak proyek. Hal ini bisa disebabkan oleh berbagai faktor seperti perubahan kebijakan bisnis, persyaratan pasar, atau pemahaman yang lebih baik tentang kebutuhan pengguna.

Penting untuk memiliki proses yang fleksibel dan adaptif untuk mengelola perubahan ini. Metodologi Agile adalah salah satu pendekatan yang efektif dalam mengelola perubahan kebutuhan, karena memungkinkan iterasi yang cepat dan penyesuaian terus-menerus selama proses pengembangan.

Stakeholder memiliki peran penting dalam menentukan keberhasilan pengumpulan kebutuhan. Tantangan di sini adalah memastikan semua stakeholder terlibat secara aktif dan komunikasi tetap terjaga. Hal ini sering kali sulit karena keterbatasan waktu stakeholder, minat yang berbeda, atau kurangnya pemahaman tentang proses teknis. Untuk mengatasi ini, penting untuk menjadwalkan pertemuan reguler, memberikan update status yang jelas, dan menggunakan alat komunikasi yang efektif. Juga, penting untuk mengidentifikasi dan melibatkan semua stakeholder kunci sejak awal proyek.

Tantangan lainnya adalah hambatan budaya dan organisasi yang dapat mempengaruhi cara kebutuhan dikumpulkan dan dikomunikasikan. Perbedaan budaya dapat mempengaruhi komunikasi, persepsi, dan ekspektasi. Organisasi dengan struktur yang kaku atau siloed dapat menghambat aliran informasi yang efektif. Untuk mengatasi hambatan ini, penting untuk mengembangkan pemahaman yang kuat tentang konteks budaya dan organisasi, dan menyesuaikan pendekatan pengumpulan kebutuhan untuk mengakomodasi perbedaan ini. Pendidikan dan pelatihan lintas budaya juga dapat sangat membantu dalam membangun pemahaman dan empati di antara tim yang beragam.

6.6 Alat dan Teknologi dalam Pengumpulan Kebutuhan

Kemajuan teknologi telah memberikan berbagai alat dan teknologi untuk memudahkan dan meningkatkan efisiensi proses pengumpulan kebutuhan, termasuk pemanfaatan kecerdasan buatan (AI). Untuk manajemen proyek dan kolaborasi, alat seperti JIRA & Confluence dari Atlassian sangat populer. JIRA efektif untuk pengelolaan kebutuhan, pelacakan isu, dan sprint planning, sementara Confluence berguna untuk

dokumentasi dan berbagi pengetahuan. Trello dan Asana menawarkan antarmuka berbasis kartu yang memudahkan pelacakan tugas dan kolaborasi tim. Dalam hal komunikasi, Microsoft Teams dan Slack memudahkan diskusi dan pertukaran informasi antara tim dan stakeholder.

Dalam aspek prototyping dan wireframing, alat seperti Sketch & Adobe XD memungkinkan desainer untuk membuat prototipe dan wireframe dengan cepat, membantu dalam visualisasi kebutuhan dan pengumpulan umpan balik. Balsamiq juga ideal untuk wireframing awal, menyediakan cara cepat untuk merancang dan berbagi ide.

Untuk analisis kebutuhan, alat seperti ReqSuite & Jama Connect dirancang khusus untuk menawarkan fitur seperti penilaian dampak perubahan dan pelacakan traceability. SurveyMonkey dan Google Forms efektif untuk pengumpulan umpan balik dan kebutuhan dari pengguna atau stakeholder melalui survei online.

Pemanfaatan AI dalam pengumpulan kebutuhan membawa inovasi signifikan. AI dapat digunakan untuk mengotomatiskan pengumpulan dan analisis data dari berbagai sumber, membantu mengidentifikasi tren dan pola. Chatbots dan asisten virtual yang didorong AI dapat digunakan untuk mengumpulkan informasi dasar kebutuhan, membebaskan tim untuk fokus pada aspek yang lebih kompleks. AI juga berperan dalam analisis sentimen dari umpan balik pengguna dan menggunakan machine learning untuk memprediksi tren masa depan dan kebutuhan pengguna, memungkinkan perencanaan yang lebih proaktif (Ahmad et al., 2021).

Integrasi alat dan teknologi modern ini, terutama AI, dalam proses pengumpulan kebutuhan meningkatkan efisiensi dan akurasi serta membuka jalan untuk pendekatan yang lebih adaptif dan responsif dalam pengembangan perangkat lunak. Hal ini mengarah pada produk yang lebih selaras dengan kebutuhan pengguna dan pasar, memastikan kesuksesan jangka panjang dalam lingkungan yang kompetitif (Pothukuchi et al., 2023).

DAFTAR PUSTAKA

Ahmad, K., Bano, M., Abdelrazek, M., Arora, C., & Grundy, J. 2021. What's up with Requirements Engineering for Artificial Intelligence Systems? *2021 IEEE 29th International Requirements Engineering Conference (RE)*, 1–12. https://doi.org/10.1109/RE51729.2021.00008

Amjad, A., Azam, F., Anwar, M. W., Butt, W. H., & Rashid, M. 2018. Event-Driven Process Chain for Modeling and Verification of Business Requirements–A Systematic Literature Review. *IEEE Access*, *6*, 9027–9048. https://doi.org/10.1109/ACCESS.2018.2791666

Deininger, M., Daly, S. R., Lee, J. C., Seifert, C. M., & Sienko, K. H. 2019. Prototyping for context: exploring stakeholder feedback based on prototype type, stakeholder group and question type. *Research in Engineering Design*, *30*(4), 453–471. https://doi.org/10.1007/s00163-019-00317-5

Dwitama, F., & Rusli, A. 2020. User stories collection via interactive chatbot to support requirements gathering. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, *18*(2), 890. https://doi.org/10.12928/telkomnika.v18i2.14866

Gokhool, O., & Nagowah, S. D. 2022. A Requirement Gathering Framework for Electronic Document Management Systems. *2022 IEEE Bombay Section Signature Conference (IBSSC)*, 1–6. https://doi.org/10.1109/IBSSC56953.2022.10037540

Lane, S., O'Raghallaigh, P., & Sammon, D. 2016. Requirements gathering: the journey. *Journal of Decision Systems*, *25*(sup1), 302–312. https://doi.org/10.1080/12460125.2016.1187390

Maiden, N. 2008. User Requirements and System Requirements. *IEEE Software*, *25*(2), 90–91. https://doi.org/10.1109/MS.2008.54

Mohy-ud-din, B., Awais, M., Sheraz, M., & Shahid, A. 2017. A Systematic Report on Issue and Challenges during Requirement Elicitation. *International Journal of*

Advanced Computer Science and Applications, *8*(11). https://doi.org/10.14569/IJACSA.2017.081145

Pfleeger, S. L., & Atlee, J. M. 2009. *Software Engineering: Theory and Practice* (4th ed.). Pearson.

Pothukuchi, A. S., Kota, L. V., & Mallikarjunaradhya, V. 2023. Impact of Generative AI on the Software Development Lifecycle (SDLC). *International Journal of Creative Research Thoughts*, *11*(8).

Pressman, R. S., & Maxim, B. R. 2020. *Software Engineering: A Practitioner's Approach* (9th ed.). McGraw-Hill Education. https://doi.org/10.1049/ic:20040411

Rezzky, E., Hendra, Tjakra, R. O., & Gunawan, A. A. S. 2021. Factors that Affect Data Gathered Using Interviews for Requirements Gathering. *2021 1st International Conference on Computer Science and Artificial Intelligence (ICCSAI), 161–166. https://doi.org/10.1109/ICCSAI 53272.2021.9609706*

Sandfreni, & Surendro, Ir. K. 2016. Requirements Engineering Model: Role Based Goal Oriented Model. *MATEC Web of Conferences*, *52*, 01004. https://doi.org/10.1051/ matecconf/20165201004

Sommerville, I. 2011. *Software Engineering* (9th ed.). Pearson Education.

Zachariah, B., & Nonyelum, O. F. 2020. A Comparative Analysis of Requirement Gathering Techniques. *IUP Journal of Computer Sciences*, *14*(2), 7–32.

BAB 7
PERANCANGAN ARSITEKTUR SISTEM DENGAN UML

Oleh Green Ferry Mandias

7.1 Arsitektur Sistem

7.1.1 Defensi

Arsitektur sistem adalah konsep yang mencakup struktur dan organisasi suatu sistem secara keseluruhan (Rozanski dan Woods, 2019). Definisi ini mencerminkan pendekatan holistik terhadap perancangan dan pembangunan sistem komputer atau informasi. Beberapa aspek penting dari definisi arsitektur sistem meliputi:

1. Struktur Sistem:
   Arsitektur sistem melibatkan pemahaman tentang bagaimana berbagai komponen atau modul dalam sistem saling berinteraksi dan diatur. Ini mencakup komponen perangkat keras (*hardware*), perangkat lunak (*software*), basis data, antarmuka pengguna, dan elemen-elemen lainnya.
2. Organisasi Sistem:
   Arsitektur sistem juga mencakup cara di mana komponen-komponen tersebut diorganisasi dan terhubung satu sama lain. Organisasi ini mencakup hubungan antar modul, alur data, dan interaksi antar bagian-bagian sistem.
3. Holistik:
   Pendekatan holistik menekankan pentingnya memahami sistem secara menyeluruh daripada hanya melihat komponen-komponen individu. Arsitektur sistem memandang sistem sebagai satu kesatuan, dan perancang arsitektur harus mempertimbangkan hubungan dan interaksi antara semua elemen.

4. Desain dan Struktur:
   Arsitektur sistem melibatkan proses perancangan dan penentuan struktur dasar sistem. Ini mencakup pemilihan model arsitektur (seperti client-server, berorientasi layanan, atau arsitektur berlapis) dan tata letak komponen-komponen utama.
5. Pertimbangan Kinerja:
   Arsitektur sistem harus mempertimbangkan kinerja sistem secara keseluruhan. Ini mencakup pemilihan teknologi, optimisasi, dan strategi untuk memastikan sistem dapat beroperasi dengan efisien dan memenuhi kebutuhan pengguna.
6. Fleksibilitas dan Skalabilitas:
   Arsitektur sistem juga harus dirancang dengan mempertimbangkan fleksibilitas untuk adaptasi dan skalabilitas untuk dapat memperluas kapasitas sistem seiring pertumbuhan dan perubahan kebutuhan.
7. Dokumentasi dan Komunikasi:
   Pentingnya dokumentasi arsitektur sistem agar dapat dipahami dan diimplementasikan oleh berbagai pihak terkait. Komunikasi yang efektif tentang arsitektur sistem sangat krusial dalam pengembangan dan pemeliharaan sistem.

   Dengan memahami dan merancang arsitektur sistem secara efektif, tim pengembangan dapat menciptakan sistem yang stabil, mudah dikelola, dan sesuai dengan kebutuhan organisasi atau pengguna akhir.

7.1.2 Komponen Utama

Arsitektur sistem melibatkan struktur organisasi dan desain keseluruhan dari suatu sistem, dan komponen utama dalam arsitektur ini menjadi fondasi yang mendukung kinerja dan fungsionalitas sistem secara keseluruhan. Pertama-tama, perangkat keras (*hardware*) adalah salah satu komponen utama arsitektur sistem. Ini mencakup semua perangkat fisik yang terlibat dalam operasi sistem, seperti server, komputer, perangkat penyimpanan data, dan perangkat jaringan. Perangkat keras memberikan dasar fisik untuk menjalankan

perangkat lunak dan memproses data. Desain perangkat keras yang efektif perlu mempertimbangkan kebutuhan sistem, termasuk kecepatan pemrosesan, kapasitas penyimpanan, dan kebutuhan jaringan untuk mendukung fungsionalitas yang diinginkan (Shaw dan Garlan, 2016).

Selain itu, perangkat lunak (*software*) juga merupakan komponen utama yang kritis dalam arsitektur sistem. Ini mencakup semua program komputer, aplikasi, sistem operasi, dan middleware yang digunakan untuk mengelola dan menjalankan sistem secara keseluruhan. Perangkat lunak memungkinkan kontrol atas perangkat keras dan menyediakan antarmuka pengguna yang memungkinkan interaksi dengan sistem. Desain perangkat lunak yang efektif melibatkan pemilihan teknologi yang sesuai, pengelolaan sumber daya, dan pengembangan aplikasi yang dapat diintegrasikan dengan baik dalam arsitektur sistem. Sistem operasi, sebagai contoh, berperan sebagai pengatur dan penyedia layanan dasar untuk perangkat keras dan aplikasi, memastikan kerja yang harmonis antara komponen-komponen tersebut. Selain itu, middleware memfasilitasi komunikasi antara aplikasi yang berbeda dan memungkinkan integrasi data yang efisien.

Komponen utama lainnya dalam arsitektur sistem adalah basis data. Basis data menyimpan, mengelola, dan memanipulasi data yang diperlukan oleh sistem. Desain basis data yang baik mempertimbangkan struktur dan relasi data, kecepatan akses, dan kebutuhan keamanan. Database management systems (DBMS) digunakan untuk mengelola akses data dan memastikan integritas data. Integrasi yang efektif antara basis data dan aplikasi memainkan peran kunci dalam mendukung fungsionalitas sistem yang tepat waktu dan akurat.

Selanjutnya, jaringan merupakan komponen utama dalam menghubungkan semua perangkat keras dan memastikan komunikasi yang efektif antara mereka. Arsitektur jaringan yang baik harus mempertimbangkan kecepatan, keandalan, dan keamanan. Komponen ini berperan dalam mendukung konektivitas dan aksesibilitas sistem, terutama dalam lingkungan yang memerlukan pertukaran data yang intensif atau operasi terdistribusi.

Dengan memahami peran dan keterkaitan komponen utama ini, perancang arsitektur sistem dapat mengembangkan desain yang kokoh dan efisien, memastikan bahwa sistem dapat beroperasi dengan optimal dan memenuhi kebutuhan pengguna atau organisasi.

7.2 Prinsip-Prinsip Desain Arsitektur

7.2.1 Modularitas

Prinsip-prinsip desain arsitektur untuk modularitas merujuk pada pendekatan sistematis dalam membangun sistem yang terdiri dari modul-modul terpisah yang dapat beroperasi secara independen namun juga berinteraksi dengan baik satu sama lain. Modularitas merupakan konsep utama dalam desain arsitektur yang bertujuan untuk meningkatkan fleksibilitas, pemeliharaan, dan pemahaman terhadap sistem secara keseluruhan. Beberapa prinsip utama untuk mencapai modularitas dalam arsitektur sistem adalah sebagai berikut (Martin, 2017):

1. Pembagian Fungsionalitas:
   Prinsip ini mengajarkan untuk membagi fungsionalitas sistem menjadi modul-modul yang berbeda. Setiap modul seharusnya memiliki tugas dan tanggung jawab yang terbatas, dengan tujuan untuk menjaga kohesi internal dan mengurangi ketergantungan antar modul. Pembagian fungsionalitas membantu dalam isolasi perubahan, sehingga perubahan pada satu modul tidak mempengaruhi modul lainnya.

2. Antarmuka Bersih:
   Antarmuka atau antar-muka yang jelas dan terdefinisi baik antar modul sangat penting. Ini melibatkan penentuan cara modul berkomunikasi satu sama lain. Antarmuka bersih membantu meminimalkan ketergantungan antar modul dan meningkatkan keseluruhan kohesi sistem.
3. Pembungkusan (*Encapsulation*):
   Prinsip ini menekankan penggunaan pembungkusan, yang artinya menyembunyikan detail internal suatu modul dan

hanya mengekspos antarmuka eksternal. Ini meningkatkan tingkat abstraksi dan melindungi detail internal modul dari modul lainnya, memungkinkan perubahan dalam satu modul tanpa mempengaruhi modul lain.

4. Kohesi Tinggi, Ketergantungan Rendah:
   Modul yang baik seharusnya memiliki kohesi tinggi, artinya elemen-elemen dalam modul tersebut berkaitan erat satu sama lain, dan ketergantungan antar modul seharusnya rendah. Kohesi tinggi meningkatkan kejelasan dan keterbacaan modul, sedangkan ketergantungan rendah meminimalkan dampak perubahan pada satu modul terhadap modul lainnya.
5. Kapsulasi Data:
   Prinsip ini menyarankan untuk membatasi akses langsung ke data oleh modul lain dan menggunakan metode-metode atau fungsi-fungsi untuk mengakses dan memanipulasi data. Ini membantu dalam menjaga keutuhan data dan mencegah modifikasi tidak sah.
6. Hubungan Locker (*Loose Coupling*):
   Hubungan locker merujuk pada tingkat ketergantungan antar modul yang rendah. Modul seharusnya tidak terlalu tergantung satu sama lain, sehingga perubahan dalam satu modul tidak menyebabkan efek domino pada modul-modul lainnya. Ini memberikan fleksibilitas dan kemudahan dalam mengganti atau mengupgrade modul tanpa mengubah seluruh sistem.

Dengan menerapkan prinsip-prinsip desain modularitas, arsitek sistem dapat menciptakan sistem yang lebih mudah dimengerti, dikembangkan, dan dikelola. Modularitas memfasilitasi pemeliharaan, pengujian, dan pengembangan yang efisien karena memungkinkan pemisahan tugas dan ketergantungan antar komponen sistem.

7.2.2 Keterbacaan

Prinsip-prinsip desain arsitektur untuk keterbacaan (*readability*) memegang peranan kunci dalam memastikan bahwa struktur dan komponen suatu sistem dapat dengan

mudah dipahami oleh pengembang, pemelihara, dan pihak-pihak terkait. Pertama-tama, penggunaan notasi dan penamaan yang konsisten adalah prinsip penting untuk meningkatkan keterbacaan arsitektur sistem. Notasi dan penamaan yang konsisten membantu menciptakan keseragaman dalam seluruh sistem, memudahkan pemahaman terhadap struktur dan hubungan antar komponen. Dengan menggunakan aturan penamaan yang jelas dan konsisten, pengembang dapat dengan cepat mengidentifikasi dan memahami peran serta fungsi setiap komponen tanpa kebingungan.

Selain itu, dokumentasi yang baik juga merupakan prinsip kunci untuk meningkatkan keterbacaan arsitektur sistem. Dokumentasi yang lengkap dan terstruktur membantu menjelaskan desain, keputusan arsitektur, dan komponen-komponen utama. Dokumentasi harus mencakup deskripsi yang rinci tentang tujuan dan fungsi masing-masing modul atau komponen, serta hubungan antar mereka. Dengan memasukkan diagram dan penjelasan yang mudah dimengerti, dokumentasi menjadi alat yang efektif dalam memfasilitasi pemahaman terhadap arsitektur sistem, baik bagi anggota tim pengembang yang baru maupun yang sudah terlibat dalam proyek.

Prinsip desain arsitektur untuk keterbacaan juga menekankan pentingnya struktur yang jelas. Komponen-komponen sistem harus diorganisir dengan logis, menciptakan aliran yang mudah dipahami oleh pembaca. Struktur yang baik membantu pengembang mengidentifikasi dan memahami hierarki serta ketergantungan antar komponen, menghindari kebingungan dan meningkatkan efisiensi pemahaman. Selain itu, penambahan komentar yang informatif juga mendukung keterbacaan. Komentar yang baik memberikan konteks dan penjelasan tambahan tentang tujuan, keputusan desain, atau hal-hal khusus yang perlu diperhatikan. Dengan cara ini, keterbacaan arsitektur sistem menjadi lebih tinggi, dan pemahaman terhadap kode menjadi lebih mendalam.

Dengan menerapkan prinsip-prinsip desain arsitektur untuk keterbacaan, tim pengembang dapat meminimalkan kesalahan, meningkatkan kolaborasi, dan mempercepat siklus pengembangan. Keterbacaan yang baik adalah investasi

berharga dalam pemeliharaan dan pengembangan sistem seiring berjalannya waktu, memastikan bahwa sistem tetap mudah dimengerti dan dapat disesuaikan dengan perubahan kebutuhan atau teknologi.

7.2.3 Skalabilitas

Prinsip-prinsip desain arsitektur untuk skalabilitas memainkan peran vital dalam memastikan bahwa suatu sistem dapat tumbuh dan beradaptasi seiring dengan meningkatnya beban atau kebutuhan. Pertama-tama, prinsip desain untuk dekomposisi menjadi modul atau mikroservis adalah kunci dalam mencapai skalabilitas yang efektif. Sistem yang terdiri dari modul atau layanan terpisah memungkinkan untuk menanggapi peningkatan permintaan dengan menyesuaikan jumlah instansi atau salinan modul yang diperlukan. Pendekatan ini memungkinkan skalabilitas horizontal, di mana kita dapat menambahkan lebih banyak sumber daya dan menanggung beban tambahan dengan mendistribusikan tugas di antara instansi-instanis tersebut. Dengan merancang sistem secara modular, tim pengembangan dapat menyesuaikan kapasitas sistem dengan lebih mudah dan efisien tanpa perlu memodifikasi seluruh sistem.

Selanjutnya, prinsip skalabilitas juga melibatkan penggunaan teknologi dan infrastruktur yang dapat diperluas dengan mudah. Pemilihan teknologi yang mendukung replikasi, penyebaran beban (load balancing), dan manajemen sumber daya dinamis sangat penting. Dalam konteks cloud computing, menggunakan layanan cloud yang dapat secara otomatis menyesuaikan kapasitas, seperti penyebaran kontainer atau serverless computing, dapat memberikan fleksibilitas dan responsivitas yang diperlukan untuk mengelola lonjakan beban dengan efisien. Dengan menggunakan teknologi ini, sistem dapat secara otomatis menanggapi peningkatan atau penurunan permintaan tanpa intervensi manual, menciptakan pengalaman pengguna yang konsisten meskipun kondisi beban kerja yang berfluktuasi.

Prinsip desain untuk meminimalkan ketergantungan (dependency) antar komponen juga menjadi faktor kunci dalam

mencapai skalabilitas. Ketergantungan yang tinggi antar komponen dapat menjadi hambatan dalam menambahkan kapasitas atau mengubah struktur sistem. Dengan meminimalkan ketergantungan, misalnya melalui desain antarmuka yang bersih dan pemisahan tugas yang jelas antar modul, tim pengembangan dapat mengisolasi perubahan dan peningkatan kapasitas tanpa mempengaruhi seluruh sistem. Prinsip ini mendukung pemahaman yang lebih baik terhadap dampak perubahan dan memastikan bahwa peningkatan kapasitas dapat dilakukan dengan cepat dan tanpa risiko kesalahan.

Melalui penerapan prinsip-prinsip desain arsitektur untuk skalabilitas, suatu sistem dapat tetap efisien dan responsif bahkan dalam menghadapi peningkatan permintaan yang signifikan. Skalabilitas yang baik adalah aspek kritis dalam perancangan sistem modern yang bertujuan untuk memberikan pengalaman pengguna yang baik dan memastikan bahwa sistem dapat berkembang sejalan dengan evolusi kebutuhan bisnis atau pengguna.

7.3 Penggunaan UML (*Unified Modeling Language*)

7.3.1 Pengantar UML

Unified Modeling Language (UML) adalah suatu bahasa pemodelan visual yang telah menjadi standar de facto dalam dunia pengembangan perangkat lunak. Diperkenalkan pertama kali pada tahun 1997 oleh Grady Booch, James Rumbaugh, dan Ivar Jacobson, UML bertujuan untuk menyediakan notasi yang seragam dan konsisten untuk merancang dan mendokumentasikan perangkat lunak. UML menawarkan seperangkat diagram dan konvensi yang memungkinkan para pengembang untuk secara efektif berkomunikasi dan merancang sistem perangkat lunak dengan tingkat abstraksi yang berbeda (OMG, 2017).

Salah satu keunggulan utama UML adalah kemampuannya untuk mendukung berbagai tahapan dalam siklus pengembangan perangkat lunak, mulai dari analisis kebutuhan hingga implementasi dan pemeliharaan. UML

menyediakan seperangkat diagram yang mencakup berbagai aspek sistem, termasuk struktur statis, perilaku dinamis, dan interaksi antar objek. Dengan menggunakan UML, tim pengembangan dapat memiliki bahasa bersama untuk berkomunikasi dan berkolaborasi tanpa memandang perbedaan latar belakang atau spesialisasi mereka.

Salah satu kegunaan utama UML adalah dalam merancang dan menggambarkan arsitektur sistem. Dalam konteks ini, UML menyediakan beberapa jenis diagram, seperti diagram kelas, diagram komponen, dan diagram sekuens, yang dapat digunakan untuk memodelkan struktur dan perilaku sistem. Diagram kelas UML, misalnya, memungkinkan pengembang untuk memodelkan kelas, atribut, dan hubungan antar kelas dalam sistem. Sementara itu, diagram sekuens memvisualisasikan interaksi dan alur kerja antar objek dalam suatu skenario tertentu. Dengan menggunakan alat ini, perancang arsitektur sistem dapat menggambarkan dengan jelas bagaimana komponen-komponen dalam sistem berinteraksi dan berkomunikasi.

Selain itu, UML juga mendukung konsep pemodelan berorientasi objek, di mana sistem dilihat sebagai kumpulan objek yang memiliki atribut dan perilaku. Ini memungkinkan perancangan sistem yang lebih intuitif dan terstruktur. UML dapat digunakan untuk mengidentifikasi dan mewakili entitas-entitas penting, serta menggambarkan hubungan dan interaksi di antara mereka.

Dalam keseluruhan, UML adalah alat yang sangat berharga dalam merancang dan menggambarkan arsitektur sistem karena memberikan notasi visual yang kaya dan fleksibel. Penggunaannya membantu meningkatkan pemahaman bersama antar anggota tim pengembangan, meminimalkan ambiguitas, dan mempercepat proses desain sistem. Oleh karena itu, pengenalan dan penerapan UML menjadi kunci dalam mencapai desain arsitektur yang efektif dan efisien.

7.3.2 Diagram UML yang Relevan

Beberapa diagram UML yang relevan untuk merancang dan menggambarkan arsitektur sistem melibatkan visualisasi berbagai aspek sistem. Berikut adalah beberapa diagram UML yang umum digunakan (Alhir, 2018):

1. Diagram Kelas:
   Deskripsi: Diagram Kelas UML adalah alat yang kuat untuk memodelkan struktur statis dari sistem. Ini mencakup kelas-kelas dalam sistem, atribut, metode, dan hubungan antar kelas.
   Penggunaan: Diagram Kelas membantu perancang arsitektur dalam merinci struktur objek dalam sistem dan hubungan antar mereka, memberikan pandangan yang kokoh terhadap elemen-elemen yang akan dibangun.
2. Diagram Komponen:
   Deskripsi: Diagram Komponen UML memodelkan komponen-komponen fisik atau logis dalam sistem, bersama dengan ketergantungan dan hubungan antar komponen tersebut.
   Penggunaan: Diagram ini berguna untuk memahami struktur fisik sistem dan bagaimana komponen-komponen tersebut berinteraksi, membantu perancang arsitektur dalam mengidentifikasi bagian-bagian sistem yang dapat diisolasi atau diimplementasikan ulang.
3. Diagram Paket:
   Deskripsi: Diagram Paket mengorganisir elemen-elemen dalam paket atau kelompok terkait dan menunjukkan ketergantungan antar paket.
   Penggunaan: Diagram ini membantu dalam merencanakan dan mengorganisir elemen-elemen sistem ke dalam kelompok-kelompok terkait, memberikan pandangan tingkat tinggi tentang struktur dan hubungan antar kelompok.
4. Diagram Sekuens:
   Deskripsi: Diagram Sekuens UML memodelkan interaksi antar objek dalam urutan waktu tertentu, menyoroti alur kerja atau skenario tertentu dalam sistem.

Penggunaan: Diagram ini membantu perancang arsitektur untuk memahami dan menggambarkan bagaimana objek-objek dalam sistem berinteraksi dan berkomunikasi satu sama lain.

5. Diagram Aktivitas:
   Deskripsi: Diagram Aktivitas UML memodelkan alur kerja atau aktivitas-aktivitas dalam sistem, menunjukkan langkah-langkah yang diambil oleh berbagai objek.
   Penggunaan: Diagram ini membantu dalam merancang dan menggambarkan proses atau aktivitas dalam sistem, memberikan pandangan yang jelas tentang alur kerja dalam sebuah fitur atau fungsi.
7. Diagram Interaksi:
   Deskripsi: Diagram Interaksi, seperti Diagram Kolaborasi atau Diagram Komunikasi, menunjukkan bagaimana objek-objek dalam sistem berinteraksi satu sama lain.
   Penggunaan: Diagram ini membantu perancang arsitektur untuk memahami dan menggambarkan hubungan dan komunikasi antar objek dalam suatu konteks tertentu.

Pemilihan diagram UML yang tepat tergantung pada kebutuhan spesifik dalam merancang arsitektur sistem. Kombinasi berbagai jenis diagram ini memberikan pandangan komprehensif terhadap berbagai aspek sistem, membantu dalam komunikasi efektif dan pemahaman yang lebih baik antara anggota tim pengembangan.

7.4 Analisis Kebutuhan

7.4.1 Identifikasi Kebutuhan

Analisis kebutuhan adalah tahap awal yang kritis dalam perancangan arsitektur sistem yang sukses. Proses ini mencakup identifikasi dan pemahaman mendalam terhadap kebutuhan fungsional dan non-fungsional dari sistem yang akan dibangun atau ditingkatkan. Identifikasi kebutuhan adalah langkah pertama dan fundamental dalam memahami apa yang diinginkan dan diperlukan oleh pengguna atau organisasi.

Berikut adalah langkah-langkah kunci dalam proses identifikasi kebutuhan system (Pressman, 2014):

1. Pengumpulan Informasi:
   Deskripsi: Tim pengembangan melakukan pengumpulan informasi dari berbagai sumber, termasuk interaksi langsung dengan pemangku kepentingan, dokumen-dokumen yang ada, dan penelitian pasar.
   Pentingnya: Pengumpulan informasi ini membantu untuk mendapatkan wawasan menyeluruh tentang lingkungan bisnis, tantangan yang dihadapi, serta kebutuhan pengguna dan pelanggan.
2. Wawancara Pemangku Kepentingan:
   Deskripsi: Melibatkan pemangku kepentingan utama seperti pengguna akhir, manajemen, dan pihak terkait lainnya dalam wawancara mendalam untuk memahami tujuan, harapan, dan kendala mereka terkait sistem.
   Pentingnya: Wawancara pemangku kepentingan membuka saluran komunikasi yang langsung dan memberikan pemahaman yang mendalam tentang perspektif dan kebutuhan pengguna.
3. Analisis Dokumen:
   Deskripsi: Meninjau dokumen-dokumen yang sudah ada, seperti laporan bisnis, analisis persyaratan lama, dan dokumentasi lainnya, untuk mendapatkan wawasan tambahan tentang kebutuhan dan konteks sistem.
   Pentingnya: Analisis dokumen membantu dalam mengidentifikasi kebutuhan yang mungkin telah diidentifikasi sebelumnya dan memastikan bahwa semua informasi yang relevan diperoleh.
4. Penciptaan Dokumen Persyaratan:
   Deskripsi: Mengorganisir dan mendokumentasikan kebutuhan sistem dengan jelas dan terstruktur dalam dokumen persyaratan.
   Pentingnya: Dokumen persyaratan menjadi panduan utama selama seluruh siklus pengembangan, menyajikan rincian yang dapat diukur dan diuji untuk memastikan pengembangan sesuai dengan kebutuhan.
5. Validasi Persyaratan:

Deskripsi: Memvalidasi kebutuhan dengan melibatkan pemangku kepentingan untuk memastikan pemahaman yang konsisten dan menghindari ambigu atau kontradiksi.
Pentingnya: Validasi persyaratan merupakan langkah penting untuk memastikan bahwa kebutuhan yang diidentifikasi benar-benar mencerminkan ekspektasi pengguna dan organisasi.

6. Prioritasi Kebutuhan:
   Deskripsi: Mengklasifikasikan dan memberikan prioritas kepada kebutuhan berdasarkan urgensi dan dampaknya terhadap tujuan bisnis dan pengguna.
   Pentingnya: Prioritasi membantu tim pengembangan untuk fokus pada pengembangan fitur dan fungsi yang paling penting dan memberikan nilai tambah paling besar.

Proses identifikasi kebutuhan sistem membentuk dasar yang kuat untuk perancangan arsitektur yang sukses. Dengan pemahaman yang mendalam terhadap kebutuhan pengguna dan organisasi, perancang arsitektur dapat mengembangkan solusi yang memenuhi harapan dan memberikan nilai tambah yang signifikan.

7.4.2 Use Case Modeling

Modeling Use Case (pemodelan use case) dalam pengembangan perangkat lunak adalah pendekatan yang efektif untuk menggambarkan interaksi antara sistem dan pengguna. Diagram *Use Case* adalah alat visual yang umum digunakan dalam *Unified Modeling Language* (UML) untuk mengilustrasikan cara pengguna berinteraksi dengan sistem (Ambler, 2014). Berikut adalah beberapa cara diagram *Use Case* membantu menggambarkan interaksi yang kompleks dan memahami kebutuhan pengguna dengan lebih baik:

1. Pemahaman Kebutuhan Pengguna:
   Diagram Use Case membantu dalam merinci kebutuhan fungsional sistem dari perspektif pengguna. Dengan mengidentifikasi use case (kasus penggunaan), yang mewakili tugas-tugas atau skenario yang dapat dilakukan

oleh pengguna, tim pengembangan dapat memahami tujuan utama pengguna.

2. Menggambarkan Fungsionalitas Sistem:
   Setiap use case dalam diagram mencerminkan satu fungsionalitas utama atau skenario penggunaan dari sistem. Dengan merinci setiap use case, diagram ini membantu tim pengembangan untuk menggambarkan fungsionalitas sistem secara komprehensif dari perspektif eksternal.
3. Menunjukkan Interaksi Pengguna:
   Use Case Modeling membantu dalam menunjukkan bagaimana pengguna berinteraksi dengan sistem untuk mencapai tujuan mereka. Setiap use case dapat dijelaskan melalui skenario atau alur kerja, memberikan pandangan yang jelas tentang langkah-langkah yang diambil oleh pengguna.
4. Mengidentifikasi Aktor:
   Use case melibatkan aktor, yang dapat berupa pengguna atau entitas lain yang berinteraksi dengan sistem. Diagram Use Case membantu dalam mengidentifikasi dan menggambarkan peran aktor serta hubungan antara aktor dan use case.
5. Menentukan Batasan Fungsionalitas:
   Dengan merinci setiap use case, diagram ini membantu menentukan batasan fungsionalitas sistem. Pengembang dapat lebih mudah memahami apa yang diharapkan dari sistem dalam konteks penggunaan nyata.
6. Komunikasi dengan Pemangku Kepentingan:
   Diagram Use Case menyediakan alat komunikasi yang efektif dengan pemangku kepentingan, termasuk pengguna akhir dan manajemen. Pemangku kepentingan dapat dengan mudah memahami dan memberikan umpan balik terhadap fungsionalitas yang diusulkan.
7. Perencanaan Pengembangan dan Pengujian:
   Dengan merinci setiap use case, tim pengembangan dapat merencanakan pengembangan dan pengujian secara lebih terperinci. Setiap use case dapat menjadi dasar untuk tahapan pengembangan dan pengujian spesifik.

8. Memfasilitasi Iterasi dan Perubahan:
   Diagram Use Case memberikan fleksibilitas dalam mengelola perubahan dan iterasi. Jika ada perubahan dalam kebutuhan pengguna, pengembang dapat dengan mudah menyesuaikan dan memperbarui diagram Use Case untuk mencerminkan perubahan tersebut.
   Dengan menggunakan Diagram Use Case, pengembang dan pemangku kepentingan dapat memiliki pandangan yang lebih baik tentang fungsionalitas sistem dan cara pengguna berinteraksi dengan sistem. Ini membantu memastikan bahwa solusi yang dikembangkan memenuhi kebutuhan pengguna dengan efektif dan efisien.

DAFTAR PUSTAKA

Alhir, S. 2018. *Business Modeling with UML: Business Patterns at Work*. John Wiley & Sons.

Ambler, S. W. 2014. *Introduction to UML 2 Use Case Diagrams*. Retrieved from http://www.agilemodeling.com/artifacts/useCaseDiagram.htm

Martin, R. C. 2017. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall

OMG. 2017. *Unified Modeling Language (UML) - Version 2.5. Object Management Group*. Retrieved from https://www.omg.org/spec/UML/2.5/

Pressman, R. S. 2014. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education.

Rozanski, N., & Woods, E. 2019. *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley

Shaw, M., & Garlan, D. 2016. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall.

BAB 8
VERIFIKASI DAN VALIDASI KEBUTUHAN DENGAN UML

Oleh Raymond Sunardi Oetama

8.1 Pendahuluan

Metode pengembangan perangkat lunak yang dipakai di sini, mengacu pada pendekatan berorientasi objek menggunakan pemodelan *Unified Modelling Language* (UML). UML merupakan bahasa visual yang menjadi standar untuk menspesifikasikan, menggambarkan, membangun, dan dokumentasi dari sistem perangkat lunak. Dalam proses pembuatan perangkat lunak, ada 7 fase yang akan dilalui selama prosesnya yaitu pertama melakukan identifikasi kemampuan sistem yang diinginkan oleh pengguna, kedua melakukan identifikasi kebutuhan di dalam progam yang akan dibuat, ketiga melakukan perancangan perangkat lunak, keempat melakukan proses pembuatan perangkat lunak, kelima melakukan pengujian perangkat lunak, keenam melakukan penyerahan, dokumentasi dan training perangkat lunak kepada pengguna, ketujuh melakukan proses perawatan dan pengembangan perangkat lunak setelah perangkat lunak dipakai oleh pengguna. Di bab 13 ini yang akan dibahas adalah fase kelima yaitu melakukan pengujian perangkat lunak yang mencakup verifikasi dan validasi perangkat lunak selama proses pembuatan perangkat lunak sedang dilakukan, sampai perangkat lunak sudah selesai dibuat. Verifikasi dan validasi kebutuhan dengan UML merupakan penerapan konsep ini pada model UML, yang mewakili struktur, perilaku, dan interaksi sistem perangkat lunak yang sedang dibuat.

Dengan semakin berkembangnya internet dan semakin banyaknya penggunaan perangkat lunak dalam kehidupan sehari-hari, maka ada kebutuhan untuk membuat perangkat

lunak berkualitas tinggi dan efisien, hal ini dapat dicapai melalui penggunaan prosedur Verifikasi dan Validasi (V&V) saat proses pengembangan dan pengujian perangkat lunak tersebut untuk menjaga dan meningkatkan kualitas perangkat lunak. Verifikasi memberikan jawaban terhadap pertanyaan apakah produk perangkat lunak yang sedang dikerjakan, dibuat dengan pendekatan yang memadai dan menunjukkan ketepatan terhadap spesifikasi dan standar yang sudah ditetapkan dalam pembuatannya, sedangkan validasi memberikan jawaban apakah perangkat lunak yang dibuat telah memadai dan menunjukkan nilai kecocokan dengan kebutuhan dan harapan pengguna dan pemangku kepentingan yang dituju saat produk perangkat lunak telah selesai dibuat. Tujuan utama penggunaan prosedur V&V adalah memberikan penilaian terkait kemampuan perangkat lunak untuk memenuhi persyaratannya, melakukan verifikasi kesesuaian perangkat lunak dengan spesifikasinya, melakukan validasi untuk memeriksa apakah perangkat lunak yang dibuat sudah memenuhi kebutuhan dan spesifikasi klien/pengguna aplikasi. V&V telah dianggap sebagai kumpulan tindakan (kerangka kerja) yang kuat dan luas terkait dengan aktivitas pengujian dan analisis di setiap tahapan siklus hidup perangkat lunak untuk mengembangkan perangkat lunak berkualitas tinggi dan efisien.

Setiap proyek pembuatan perangkat lunak harus melakukan verifikasi dan validasi perangkat lunak yang diproduksinya. V&V secara luas menguji dan menganalisis perangkat lunak untuk menetapkan kepastian perangkat lunak terhadap semua fungsi sistem dan terhadap antarmuka pengguna, perangkat keras, serta perangkat lunak, sudah dibuat sesuai dengan spefisikasi yang sudah ditetapkan dan dapat digunakan dengan benar, serta mengukur keandalan dan kualitasnya, yang pada akhirnya memungkinkan pencapaian harapan pengguna saat memakai perangkat lunak tersebut. Dalam dunia kompetitif modern yang terkait dengan perangkat lunak, pengembang perangkat lunak harus membuatkan produk berkualitas bisa diselesaikan tepat waktu, pengembang juga harus memverifikasi bahwa perangkat lunak telah berfungsi dengan baik dan memvalidasi produk untuk setiap kebutuhan

klien. V&V menjadi fase penting yang perlu dilakukan oleh pengembang perangkat lunak, meningkatkan wawasan manajemen proyek terhadap resiko proses dan produk, mendukung proses siklus hidup perangkat lunak untuk memastikan kepatuhan terhadap jadwal program, kinerja serta persyaratan anggaran. Proses V&V memberikan evaluasi terkait produk perangkat lunak serta proses selama siklus hidup perangkat lunak. Evaluasi tersebut menunjukkan apakah persyaratan sistem dan perangkat lunak telah tepat, dapat diuji, akurat, lengkap, serta konsisten.

Perkembangan cepat dalam V&V perangkat lunak sangat penting dalam mengembangkan pendekatan dan alat untuk mengidentifikasi kemungkinan bug yang terjadi secara bersamaan dan oleh karena itu memverifikasi kebenaran perangkat lunak. Bug adalah sebuah kesalahan atau penulisan kode program yang tidak tepat, yang menyebabkan aplikasi atau program tidak berfungsi dengan baik atau tidak berjalan sebagaimana mestinya. Ada banyak kondisi terkait kesalahan perangkat lunak yang terbukti sangat merugikan, memalukan, dan seringkali berbahaya. V&V memfasilitasi deteksi dini dan koreksi kesalahan perangkat lunak. Verifikasi melibatkan pemeriksaan bahwa suatu produk memenuhi spesifikasi atau persyaratan tertentu selama proses pembuatannya. Banyak pengembang perangkat lunak menggunakan daftar periksa atau serangkaian persyaratan untuk membuat fitur dan kemampuan spesifik untuk suatu teknologi. Selama proses pembuatan perangkat lunak, dapat dilakukan pemeriksaan pekerjaan mereka berdasarkan daftar ini untuk memastikan kode yang digunakan memenuhi harapan. Jika ada kesalahan dalam kode, maka dapat memperbaikinya di awal proses produksi. Beberapa keuntungan menggunakan verifikasi: membantu menghemat waktu dengan mengetahui cacat pada tahap awal pengembangan, memastikan perangkat lunak bebas dari cacat dan mematuhi standar pada setiap tahap proses pengembangan, membantu tim pengembangan dalam membangun perangkat lunak sesuai spesifikasi klien. Validasi akan menguji kelengkapan suatu produk dan mengukur apakah produk tersebut memenuhi kebutuhan pemangku kepentingan atau

pelanggan yang memintanya. Dengan menggunakan kebutuhan pemangku kepentingan yang sudah ada, memungkinkan membuat serangkaian pengujian otomatis yang menjalankan perangkat lunak dan mengukur kinerja perangkat lunak tersebut, lalu memeriksa perangkat lunak tersebut tentang hasil pengujiannya. Dengan cara ini, pemrogram dapat memahami bagaimana suatu perangkat lunak mungkin gagal dan menganalisis kode untuk memperbaikinya. Memvalidasi kinerja suatu produk sering kali berarti mengujinya, di sini penguji seolah-olah menjadi pengguna akhir, yang biasanya adalah pelanggan. Jenis pengujian ini membantu untuk memeriksa apa hasil yang mungkin dilihat pengguna saat menggunakan perangkat lunak tersebut, dan memberi penguji perangkat lunak gambaran yang lebih baik tentang potensi kesalahan atau bug dalam kode. Berikut beberapa keuntungan menggunakan validasi: memastikan desain akhir memenuhi persyaratan pelanggan, mengizinkan tim menemukan dan memperbaiki kesalahan apa pun yang mereka buat selama proses pengembangan, bertindak sebagai kontrol kualitas produk. Perlindungan optimal terhadap kesalahan ini adalah untuk memvalidasi kebenaran dan meningkatkan kualitas keseluruhan terkait perangkat lunak sebelum digunakan. Sebuah produk yang memenuhi kebutuhan pelanggan sejak awal diperkenalkan ke pasar kemungkinan besar akan mendapatkan ulasan positif, berkinerja baik, dan meningkatkan reputasi perusahaan. Pada akhirnya, hal ini dapat meningkatkan penjualan dan meningkatkan pendapatan perusahaan.

8.2 Definisi Umum Untuk V&V

Dalam pembuatan perangkat lunak berdasarkan kebutuhan atau permintaan pengguna, diperlukan proses verifikasi dan validasi perangkat lunak tersebut untuk menjamin kualitas perangkat lunak yang dihasilkan. Verifikasi berkaitan dengan kebenaran proses pembuatan perangkat lunak, untuk memastikan perangkat lunak yang dibuat sesuai dengan spesifikasi yang telah ditentukan pada proses perencanaan. Verifikasi digunakan untuk menentukan apakah

perangkat lunak yang terkait dengan siklus hidup pengembangan sistem fase tertentu memenuhi persyaratan yang dikembangkan sepanjang tahap sebelumnya atau tidak. Metode verifikasi dipakai untuk mengidentifikasi kesalahan atau menemukan kesalahan perangkat lunak yang mengakibatkan kegagalan. Validasi berkaitan dengan kebenaran produk. Validasi merupakan proses untuk mengevaluasi perangkat lunak di bagian akhir pembuatannya untuk memastikan bahwa perangkat lunak tersebut bebas dari kegagalan serta memenuhi persyaratannya, juga untuk memastikan bahwa produk memenuhi persyaratan klien, serta spesifikasinya telah memadai dan perangkat lunak terlindung dari akses tidak sah. Dengan demikian, validasi memastikan bahwa perangkat lunak dapat berfungsi dengan baik untuk data masukan yang benar serta memberikan pesan kesalahan untuk data masukan yang salah.

Berikut ini diberikan berbagai definisi terkait istilah verifikasi dan validasi dalam literatur dan standar terkait:

1. Dari IEEE-1012-2012 (IEEE Computer Society, 2012) (IEEE, 2004):
   a. Verifikasi didefinisikan sebagai prosedur untuk mengevaluasi komponen atau sistem untuk menilai apakah produk yang terkait dengan tahap tertentu memenuhi kondisi di awal tahap.
   b. Validasi didefinisikan sebagai proses untuk mengevaluasi komponen atau sistem pada akhir atau seluruh proses pengembangan untuk menentukan apakah memenuhi persyaratan tertentu.
2. ISO/IEC 15288 (2015) memberikan definisi berikut terkait dengan verifikasi dan validasi sistem (Standard, 2008):
   a. Verifikasi: dapat didefinisikan sebagai serangkaian aktivitas yang membandingkan produk yang berkaitan dengan siklus hidup sistem dengan karakteristik produk yang diperlukan. Hal ini mungkin melibatkan, namun tidak terbatas pada persyaratan tertentu, deskripsi desain, serta sistem itu sendiri.
   b. Validasi: dapat didefinisikan sebagai serangkaian aktivitas yang menjamin bahwa sistem memiliki

kemampuan untuk mencapai tujuan, sasaran, serta sesuai tujuan penggunaan.

3. Buku Panduan INCOSE SE, v4, mengembangkan definisi terkait verifikasi dan validasi yang digunakan dalam ISO/IEC 15288 serta menambahkan catatan tertentu sebagai berikut (Walden, 2015):
   a. Verifikasi: ini didefinisikan sebagai serangkaian kegiatan yang membandingkan elemen sistem atau sistem terhadap karakteristik yang diperlukan.
   b. Validasi: ini dapat didefinisikan sebagai serangkaian aktivitas yang menjamin bahwa sistem memiliki kemampuan untuk mencapai tujuan, sasaran, dan tujuan penggunaannya (memenuhi persyaratan pemangku kepentingan) dalam lingkungan operasional yang diinginkan.
4. ISO/IEC/IEEE 29148:2011 (revisi 2018) mengulangi definisi ISO/IEC 15288 terkait verifikasi dan validasi, dan memberikan definisi persyaratan verifikasi dan validasi berikut (Altaie, Alsarraj and Al-Bayati, 2020):
   a. Verifikasi persyaratan: didefinisikan sebagai serangkaian persyaratan yang diindikasikan untuk memastikan karakteristik yang berkaitan dengan persyaratan optimal telah dilakukan.
   b. Validasi persyaratan: mendefinisikan sistem yang memadai yang direncanakan melalui pemangku kepentingan.

Verifikasi akan memastikan bahwa perangkat lunak telah selaras dengan persyaratannya, memenuhi kebutuhan tertentu, kinerja, serta kelengkapan. Hal ini mengupayakan terpenuhinya konsistensi, kebenaran terkait penerjemahan program selain kebenaran perilaku. Validasi digunakan untuk mengetahui kebenaran perangkat lunak pada setiap tahapan yang terkait dengan siklus pengembangan.

8.3 Peran V&V Dalam Siklus Hidup Pengembangan Sistem Perangkat Lunak

Peran V&V terhadap masing-masing produk harus dinilai melalui landasan proyek per proyek. Penilaian tersebut akan dipengaruhi oleh kekritisan produk, kendalanya, serta kompleksitasnya. Secara umum, tujuan fungsi V&V adalah memastikan bahwa produk memenuhi kebutuhan pengguna. Oleh karena itu, segala sesuatu dalam spesifikasi dan persyaratan produk harus sesuai dengan aktivitas V&V tertentu. Namun, untuk membatasi ruang lingkup modul tersebut, pendekatan V&V yang ditentukan akan berkonsentrasi pada bagian fungsional dan kinerja terkait dengan spesifikasi dan persyaratan dalam hal keamanan, pemeliharaan, portabilitas, keselamatan, kegunaan, serta kemudahan servis. Verifikasi bertujuan untuk "membangun sistem yang benar", dan validasi bertujuan untuk "membangun sistem yang tepat". Oleh karena itu, verifikasi memeriksa masalah seperti jaminan bahwa pengetahuan sistem telah ditentukan secara memadai, sedangkan validasi memeriksa proses untuk memastikan sistem membuat keputusan yang benar. V&V sangat penting dalam implementasi dan pengembangan sistem berbasis kasus. Jika sistem tidak terverifikasi, mungkin ada kesalahan dalam representasi kasus. Dalam kasus ketika sistem belum divalidasi, maka sistem tersebut mungkin tidak menghasilkan kualitas keputusan yang diperlukan. Sangat mungkin suatu produk lolos saat diverifikasi tetapi gagal saat divalidasi. Hal ini dapat terjadi ketika, katakanlah, suatu produk dibuat sesuai spesifikasi namun spesifikasi itu sendiri gagal memenuhi kebutuhan pengguna.

V&V dalam siklus hidup pengembangan sistem perangkat lunak akan disajikan pada bagian ini. Khususnya, apa yang diperiksa sehubungan dengan seluruh fase siklus hidup serta siapa yang melakukan pemeriksaan tersebut. Juga pendekatan yang digunakan untuk melakukan pengecekan. Selain itu, V&V perangkat lunak menggunakan pendekatan tinjauan, pengujian, dan analisis untuk menentukan apakah sistem perangkat lunak serta produk perantaranya memenuhi tuntutan. Tuntutan

tersebut mencakup atribut kualitas serta kemampuan fungsional. Selain itu, tugas utama V&V adalah mengelola risiko proyek melalui pemantauan dan deteksi kesalahan selama proses pemeliharaan dan pengembangan. Karena kenyataan bahwa tidak praktis untuk mendeteksi dan menyelesaikan semua kesalahan di awal siklus hidup suatu proyek, maka tugas kontraktor V&V adalah mengidentifikasi kesalahan sedini mungkin serta melacak kemajuan menuju penyelesaian. Selain itu, verifikasi memastikan bahwa sistem (personel, perangkat keras, perangkat lunak, dan dokumentasi) sesuai dengan proses dan standar organisasi, berdasarkan pendekatan tinjauan atau non-eksekusi. Selanjutnya, validasi secara fisik memastikan bahwa sistem beroperasi berdasarkan rencana melalui pelaksanaan fungsi sistem melalui serangkaian pengujian yang dapat dinilai dan diamati.

Siklus hidup produk memiliki definisi yang menentukan perilaku sistem yang diantisipasi. Setelah itu, solusi akan dikembangkan, dibangun, dan diuji untuk membandingkan perilaku dengan gambaran awal. Apabila lolos maka akan dimanfaatkan, namun bila tidak lolos maka akan dikerjakan ulang hingga lolos. Selain itu, artefak siklus hidup merupakan produk sampingan yang terkait dengan evolusi tersebut. Artefak tersebut mewakili sistem pada fase tertentu dari siklus hidup seperti persyaratan, desain, kode, dll. Persyaratan mewakili perilaku, desain mewakili solusi, sedangkan kode sumber mewakili implementasi, juga pengujian mewakili argumen kualifikasi sehubungan dengan penerapan. V&V dapat dianggap sebagai sebuah proses tersendiri, juga memiliki siklus hidupnya sendiri. Siklus hidup V&V akan dijalankan secara paralel dengan pengembangan. Misalnya, karena perilaku telah ditentukan dan produk sampingan dibuat (spesifikasi kebutuhan), V&V akan melakukan analisis kebutuhan. Atas dasar evaluasi mereka, proses V&V akan memperoleh pemahaman terkait perilaku sistem.

Berikut ini beberapa saran penting yang perlu dipertimbangkan agar sukses melakukan proses Verifikasi dan Validasi:

1. Tetapkan tujuan spesifik untuk proses verifikasi dan validasi dengan jelas agar dapat mencapai hasil yang diinginkan.
2. Gunakan metode verifikasi yang sesuai seperti inspeksi, pengujian, atau simulasi untuk menilai kebenaran dan kelengkapan sistem secara efektif.
3. Kembangkan rencana terperinci yang menguraikan ruang lingkup, jadwal, sumber daya, dan kriteria untuk mengevaluasi kinerja sistem.
4. Melaksanakan pengujian yang ketat dan menyeluruh untuk mengidentifikasi cacat, kesalahan, atau inkonsistensi dalam fungsi sistem dan memastikan sistem memenuhi semua persyaratan yang ditentukan.
5. Libatkan pemangku kepentingan yang relevan di seluruh proses untuk mengumpulkan umpan balik, mengatasi kekhawatiran, dan memvalidasi bahwa sistem memenuhi harapan mereka.
6. Mendokumentasikan catatan yang tepat mengenai semua aktivitas verifikasi dan validasi yang dilakukan, termasuk masalah apa pun yang teridentifikasi dan penyelesaiannya, untuk memudahkan melacak hasil dan memastikan ketertelusuran dan akuntabilitas.

Sebuah produk yang memenuhi kebutuhan pelanggan sejak awal diperkenalkan ke pasar kemungkinan besar akan mendapatkan ulasan positif, berkinerja baik, dan meningkatkan reputasi perusahaan. Pada akhirnya, hal ini dapat meningkatkan penjualan dan meningkatkan pendapatan perusahaan.

8.4 Detail Perbedaan Pengujian Antara Verifikasi Dan Validasi

Berikut ini diberikan detail perbedaan pengujian perangkat lunak antara Verifikasi dan Validasi:

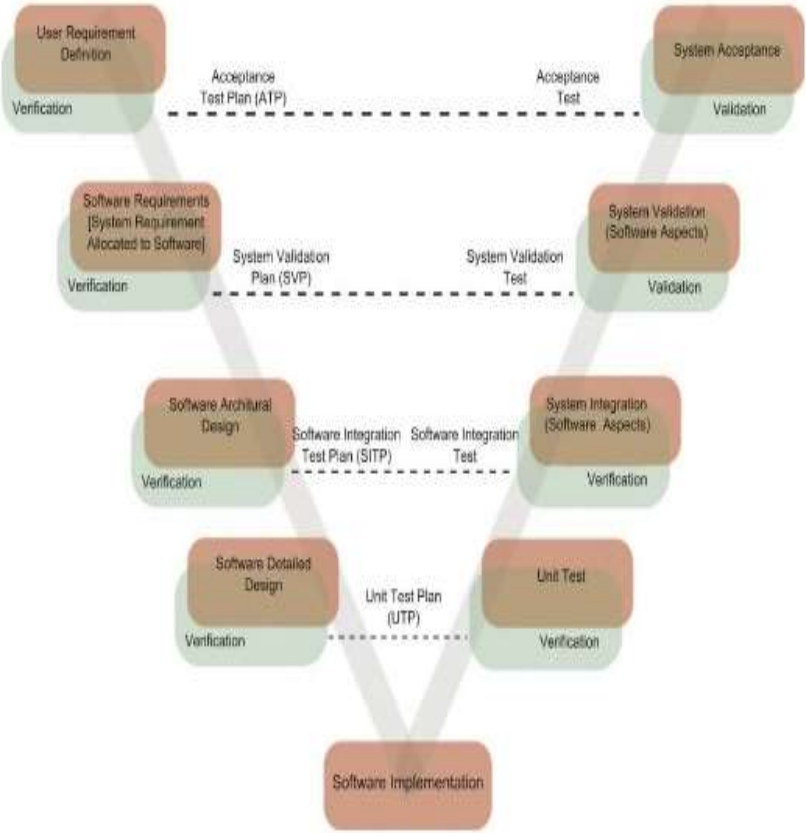
Tabel 8.1. Perbedaan pengujian antara Validasi dan Verifikasi

Parameter	Verifikasi	Validasi
Waktu Pelaksanaan	Dilakukan pada awal siklus pengembangan perangkat lunak, di tingkat komponen, modul, dan sistem.	Dilakukan setelah perangkat lunak sudah selesai dibuat di tingkat sistem.
Fokus kegiatan	Pengujian verifikasi perangkat lunak terutama berfokus pada dokumen spesifikasi, desain tabel database, dokumen desain arsitektur perangkat lunak, diagram ER (*Entity Relationship* Diagram), matriks ketertelusuran, kasus dan skenario pengujian, dan banyak lagi.	Validasi dalam pengujian perangkat lunak berfokus pada perangkat lunak akhir yang akan digunakan pelanggan.
Jenis pekerjaan pemeriksaan	Praktik statis untuk memverifikasi apakah perangkat lunak memenuhi semua spesifikasi yang telah ditentukan dalam tahap pengembangan.	Praktik dinamis dalam menguji perangkat lunak yang sudah selesai dibuat untuk memastikan perangkat lunak tersebut memenuhi persyaratan pelanggan.
Eksekusi	Verifikasi tidak	Validasi

Parameter	Verifikasi	Validasi
kode	memerlukan eksekusi kode (sebab saat itu perangkat lunak belum selesai dibuat).	memerlukan eksekusi kode (saat perangkat lunak sudah selesai dibuat).
Evaluasi	Metode pengujian perangkat lunak ini mencakup verifikasi manusia terhadap aset penting.	Metode pengujian perangkat lunak ini mencakup pemeriksaan dan validasi perangkat lunak yang melibatkan manusia dan mesin.
Deteksi bug	Verifikasi mendeteksi bug pada awal setiap tahap pengembangan perangkat lunak.	Validasi hanya mendeteksi bug yang terlewatkan selama proses verifikasi.
Pelaksana kegiatan pengujian	Pengembang dan penguji mutu (QA) melakukan pengujian verifikasi untuk memastikan produk sesuai dengan spesifikasi dan standar yang telah direncanakan sebelumnya.	Validasi terutama dilakukan oleh pakar Jaminan Kualitas, juga bisa dibantu oleh tim penguji mutu (QA) yang memeriksa perangkat lunak secara menyeluruh untuk memastikan perangkat lunak berfungsi sebagaimana mestinya.
Teknik yang dipakai	Verifikasi perangkat lunak menggunakan teknik seperti inspeksi,	Proses validasi melibatkan teknik pengujian perangkat

Parameter	Verifikasi	Validasi
	penelusuran, peninjauan produk dan dokumen.	lunak yang berbeda seperti pengujian unit, pengujian regresi, pengujian integrasi, dan pengujian non-fungsional.
Peralatan Pendukung yang perlu digunakan	Verifikasi dalam pengujian perangkat lunak tidak memerlukan platform, perangkat, sistem operasi, atau browser apa pun.	Validasi perangkat lunak menuntut penggunaan perangkat, browser, dan sistem operasi.

Gambar di bawah ini memberikan gambaran yang lebih jelas tentang proses verifikasi dan validasi (V&V) dan bagaimana kedua proses ini bekerja bersama-sama dalam siklus hidup pengembangan sistem perangkat lunak untuk mencapai tujuan bersama:



Gambar 8.1. The V-model of software development
Sumber (Karmakar *et al.*, 2023)

8.5 Alat Dan Pendekatan V&V

8.5.1 Alat dan Pendekatan V&V Di Bidang Modeling

Pada tahun 2003, Engels dkk mempresentasikan metode untuk memverifikasi karakteristik model UML bahasa pemodelan terpadu melalui penerapan pemeriksa model FDR Kegagalan-divergensi-Penyempurnaan dan memvalidasi model pengujian. Jadi, pengujian nanti dapat ditingkatkan pada level model melalui penyediaan metode derivasi implementasi dari model sistem UML serta model pengujiannya, dengan mempertimbangkan strategi pengujian tertentu. Hasilnya menunjukkan bahwa rangkaian alat memungkinkan verifikasi

dan validasi properti melalui penyediaan jenis terjemahan parsial yang telah ditentukan sebelumnya untuk memverifikasi dan menguji strategi validasi (Engels *et al.*, 2003).

Pada tahun 2010, Kezadri dan Pantel memaparkan unsur-unsur utama terkait ontologi yang memformalkan bagian mengenai pengetahuan tentang pemodelan perilaku serta pendekatan V&V terkait. Ini merangkum entitas, konsep, dan objek yang umurnya diindikasikan ada dalam bidang minat tersebut serta hubungan di antara mereka. Studi ini menyarankan klasifikasi terkait dengan berbagai formalisme pemodelan, representasi terkait kemungkinan pendekatan V&V. Sistem ditentukan menggunakan banyak tampilan sesuai dengan banyak bahasa pemodelan, dan properti juga dapat dievaluasi dengan V&V (Kezadri and Pantel, 2012).

Pada tahun 2013, Delmas et al mendefinisikan proses umum verifikasi dan validasi untuk desain model dan pembuatan instance yang dipusatkan pada diagram kelas UML dan batasan deklaratifnya. Penelitian ini mengilustrasikan cara pendekatan formal dapat digunakan selama berbagai tahapan prosedur MDE (*Model-Driven Engineering*), mulai dari formalisasi masalah hingga sintesis solusi, dan verifikasi solusi (Delmas, Pires and Polacsek, 2013).

Pada tahun 2016, dibuat sebuah makalah dengan tujuan utama terletak pada membuat para ahli teknik Sistem mampu menciptakan DSML Bahasa Pemodelan Spesifik Domain yang cukup formal dan mampu memodelkan sistem yang memastikan aktivitas V&V-nya. Jadi, pendekatan ini bertujuan untuk secara formal mendefinisikan dan mempromosikan gagasan semantik dinamis untuk DSML, memungkinkan kemampuan untuk mensimulasikan model yang dibuat dengan menggunakan teknik simulasi dan pemeriksaan model secara independen dari aturan dan mekanisme transformasi. Makalah yang dibuat tersebut menyediakan metode yang dilengkapi alat yang disebut "xviCore" yang memungkinkan para insinyur sistem untuk membuat DSML formal yang cukup dan mampu mencapai tujuan V&V dan membuat model yang dapat disimulasikan dan diambil sebagai masukan untuk proses pembuktian properti (Nastov *et al.*, 2016).

Pada tahun 2017, Nastov dkk memberikan model untuk menggabungkan empat strategi utama yang saling melengkapi untuk verifikasi dan validasi yaitu pembuktian formal, pemodelan terbimbing, simulasi, dan penilaian model. Meskipun pelaksanaan dan kombinasi strategi-strategi ini memakan waktu, bermasalah, dan mahal, penelitian ini menyajikan alat yang dilengkapi untuk memudahkan pelaksanaan dan kombinasi strategi V&V dalam alat yang disebut xviCore dan memperkenalkan tingkat kepercayaan yang dapat diterima dalam pengambilan keputusan bagi pelanggan. Alat xviCore adalah alternatif yang menarik untuk mengarahkan desainer agar tetap mandiri dan terorganisir dengan baik, membuat model dengan upaya yang lebih rendah, dan mendapatkan berbagai hasil dari strategi V&V (Nastov *et al.*, 2017). Pada tahun yang sama, Gogolla dkk memperkenalkan dan memberikan kasus penggunaan yang diperlukan untuk memodelkan perangkat lunak menggunakan UML dan OCL dalam banyak aspek yaitu verifikasi model, validasi, dan eksplorasi untuk membantu insinyur perangkat lunak menemukan cacat dan kesalahan dalam deskripsi model dan selanjutnya meningkatkan kualitas model. Eksperimen diterapkan delapan use case dan ditampilkan validator dengan model USE. Pendekatan ini sangat berharga ketika menggambar model dari tahap awal pengembangan hingga verifikasi properti model pada tahap pengujian (Gogolla, Hilken and Doan, 2018).

Pada tahun 2018, Baduel dkk mempresentasikan pendekatan yang terkait dengan verifikasi dan validasi model *Systems Modeling Language* (Sys-ML) dari sudut pandang industri. Pendekatan ini menggunakan *Object Constraint Language* (OCL) untuk verifikasi model dan memanfaatkan *Bombardier Transportation* (BT) untuk mencapai hasil V&V yang memadai dan mempercepat proses pengembangan sistem dengan waktu yang lebih sedikit untuk validasi dan pengujian sistem (Baduel *et al.*, 2018).

Pada tahun 2019, Schumann dan Popstojanov telah mempresentasikan model V&V yang dirancang khusus untuk Rekayasa Perangkat Lunak Berbasis Model (MBSwE), yang mencerminkan keterkaitan antara variasi alat dan artefak di

satu sisi, dan berbagai tugas V&V (seperti analisis, peninjauan, dan pengujian) di sisi lain, memisahkan semua tugas V&V ke dalam tiga tingkat: V&V tingkat model, V&V tingkat perangkat keras, dan V&V tingkat kode. V&V dalam Rekayasa Perangkat Lunak Berbasis Model tetap merupakan tugas yang berharga, namun rumit, terutama dalam kasus di mana beberapa bagian perangkat lunak dihasilkan secara otomatis dari model, sedangkan yang lain ditulis secara manual (Schumann and Goseva-Popstojanova, 2019).

8.5.2 Alat dan Pendekatan V&V Di Bidang Coding dan Aplikasi Industri

Pada tahun 2006, Sherriff dan Williams menyajikan model parametrik yang menerapkan catatan persisten terkait praktik (V&V) yang digunakan program untuk memperkirakan kepadatan kerusakan program. Selain itu, catatan persisten terkait praktik V&V telah dicatat sebagai sertifikat yang telah dicatat secara otomatis serta dikelola dengan kode. Selain itu, model ini memungkinkan pengembang untuk menangani upaya yang telah dilakukan dalam V&V di mana semua pengembang mungkin melihat langkah-langkah yang diambil untuk memastikan keandalan terkait dengan potongan kode dan memperlakukannya sesuai dengan itu (Sherriff and Williams, 2006).

Pada tahun 2010, Feldt dkk mempresentasikan dua studi kasus industri dari perusahaan-perusahaan di industri luar angkasa Eropa yang telah mengikuti standar Kerjasama Eropa untuk Standardisasi Luar Angkasa (ECSS) dalam berbagai aktivitas V&V. Studi-studi yang telah dilaporkan dalam makalah ini berfokus pada cara standar ECSS digunakan oleh perusahaan, cara dan sejauh mana standar tersebut mempengaruhi prosedur mereka, dan pada akhirnya, mereka telah membahas cara-cara yang potensial untuk meneruskannya sehingga dapat mencapai tujuan yang diharapkan, mencapai tujuan penciptaan kerangka kegiatan yang lebih hemat biaya untuk verifikasi dan validasi industri luar angkasa (Feldt *et al.*, 2010).

Pada tahun 2018, Sen dkk memaparkan gambaran umum terkait Certus yang merupakan pusat penelitian V&V perangkat

lunak yang didorong melalui industri perangkat lunak. Certus juga mendirikan kumpulan pengetahuan, alat, dan pendekatan untuk V&V terkait sistem perangkat lunak di sektor publik dan swasta Norwegia. Selain itu, karya ini menyajikan struktur organisasi Certus dan menggambarkan bagaimana kesehatan dan kehidupan Certus telah direncanakan serta sering menilai pusat inovasi berbasis penelitian. Kesehatan Certus dinilai berdasarkan banyak kriteria seperti kesetaraan gender, upaya standardisasi, publikasi, indeks inovasi, serta kerja sama internasional (Sen, Marijan and Gotlieb, 2018).

Pada tahun 2019, Bondarev dkk menangani klasifikasi terkait pendekatan verifikasi perangkat lunak (*software*). Artikel terkait pendekatan verifikasi statis saat ini telah diterapkan, karakteristik terkait pendekatan telah diperiksa, juga telah dicapai studi tentang deteksi terkait ketergantungan dalam kerangka terkait pendekatan interpretasi abstrak. Selain itu, artikel ini berisi ikhtisar terkait klasifikasi serta pendekatan verifikasi perangkat lunak dinamis. Pendekatan dinamis memfasilitasi penentuan kesalahan yang terjadi saat memulai program. Selain itu, ini merupakan pendekatan pengujian yang lebih produktif dan modern yang dapat mendeteksi sejumlah besar kerentanan dalam kode program, daripada menggunakan pendekatan statis yang terkait dengan pengujian perangkat lunak (Bondarev, Chudinov and Prokhorov, 2019).

8.5.3 Alat dan Pendekatan V&V Di Bidang Kualitas Perangkat Lunak

Pada tahun 2006, Runeson et al memberikan analisis terhadap karya empiris saat ini yang menunjukkan tidak ada jawaban jelas atas pertanyaan terkait pendekatan deteksi cacat apa yang harus dipilih. Sehubungan dengan cacat persyaratan, biaya yang berkaitan dengan inspeksi persyaratan lebih rendah dibandingkan dengan melakukan persyaratan yang tidak memadai. Berkenaan dengan cacat spesifikasi desain, terdapat penelitian tertentu yang menunjukkan bahwa inspeksi lebih efektif pada tahap tersebut. Berkenaan dengan kode etik, pengujian struktural atau fungsional dinilai lebih efektif dibandingkan dengan inspeksi di sebagian besar penelitian.

Efisiensi verifikasi masih rendah; pengulas hanya menemukan (25-50)% cacat dengan menggunakan inspeksi, juga penguji menunjukkan (30-60)% dengan menggunakan pengujian (Runeson *et al.*, 2006).

Pada tahun 2008, Drusinsky et al memperkenalkan model yang memungkinkan para insinyur perangkat lunak dalam memilih pendekatan yang memadai sehubungan dengan tugas-tugas V&V formal. Model yang menyediakan ruang tradeoff 3D melibatkan cakupan dan biaya. Oleh karena itu, sangat penting dalam memilih pendekatan V&V formal yang efektif cakupan dan hemat biaya dalam dimensi spesifikasi/validasi (Drusinsky, Michael and Shing, 2008).

Pada tahun 2009, Monteiro dkk memberikan pelaksanaan dan analisis penerapan aktivitas V&V dalam CMMI (*Capability Maturity Model Integration*) dan ML-2 (*Maturity Level* 2) yang menghasilkan peningkatan atau penurunan siklus hidup produk. Selanjutnya, analisis dampak dilakukan melalui perbandingan keadaan siklus hidup produk sebelum dan sesudah diperkenalkannya V&V *Process Areas* (PA) berdasarkan hasil yang dicapai. Dalam hal hasil belum memadai, beberapa variasi penggabungan praktik V&V ke dalam CMMI ML-2 dilakukan hingga mencapai hasil yang optimal (Monteiro, Machado and Kazman, 2009).

Pada tahun 2012, Pasareanu dan Bobaru memperkenalkan pendekatan pembelajaran yang semakin banyak digunakan untuk meningkatkan aktivitas V&V perangkat lunak. Misalnya, pendekatan pembelajaran automata digunakan untuk mengekstrak model perilaku yang terkait dengan sistem perangkat lunak. Penelitian ini melibatkan lima presentasi, dimana 4 makalah awal membahas pembelajaran automata serta menyajikan banyak pendekatan untuk mempelajari banyak jenis yang terkait dengan automata. Studi terakhir memiliki konsentrasi yang berbeda, karena mempelajari hubungan antara pengujian otomatis dan pembelajaran mesin (Păsăreanu and Bobaru, 2012).

Pada tahun 2018, Cheng dkk memberikan metode praktis dan dasar pemikiran statistik yang efektif untuk mengevaluasi kualitas produk secara keseluruhan melalui aktivitas V&V.

Selain itu, metode yang diusulkan berguna untuk memperoleh rencana yang sesuai untuk memprediksi persyaratan keandalan produk melalui tabel dan persamaan. Keandalan harus meningkatkan ambang batas dengan nilai prediksi kepercayaan statistik dan memberikan visi untuk memprediksi parameter kriteria penerimaan produk dan kelulusannya. Hasil dan contoh menunjukkan bagaimana mencapai nilai penerimaan tingkat tinggi (Cheng *et al.*, 2019).

Pada tahun 2019, pemilihan pendekatan V&V untuk mengakomodasi fitur kualitas ISO-25010. Oleh karena itu, penulis mengumpulkan serangkaian pendekatan V&V dan menunjukkan kepada para ahli untuk menilai bagaimana setiap pendekatan menerapkan fitur ISO-25010. Hasilnya menyebutkan dua puluh praktik terbaik yang dilakukan lebih dari satu pakar dapat diimplementasikan secara otomatis atau semi otomatis untuk meningkatkan kualitas perangkat lunak termasuk berbagai jenis pengujian: Fuzz, Penetration, Stress, dan Model Checking (Mendoza *et al.*, 2019). Pada tahun yang sama, Mobin dkk, makalah ini memperkenalkan model untuk meningkatkan perencanaan aktivitas V&V dan desain produk untuk meningkatkan keandalan dan memberikan informasi penting kepada pengembang tentang produk. Kegiatan perencanaan V&V yang disarankan mempertimbangkan kemampuan deteksi, prioritas mode kegagalan, urutan pelaksanaan aktivitas V&V, efisiensi masing-masing aktivitas, dan konsekuensi dalam deteksi kegagalan. Tujuan dari model ini adalah memaksimalkan keandalan berdasarkan pemilihan rangkaian aktivitas V&V yang optimal dan pengurutannya berdasarkan penjadwalan job shop (Mobin *et al.*, 2019).

8.5.4 Alat Pengujian Verifikasi dan Validasi

Proses pengujian perangkat lunak dilakukan untuk memastikan bahwa perangkat lunak yang dikembangkan sudah berjalan dengan semestinya.

Berikut ini metode pengujian yang biasa digunakan untuk Verifikasi Perangkat Lunak. Mula-mula, tim penguji harus memastikan dahulu bahwa semua hal berikut ini memenuhi standar yang berlaku:

1. Memverifikasi persyaratan: Tim proyek mengonfirmasi bahwa semua item dalam dokumen persyaratan sudah benar berdasarkan masukan dari bisnis. Pelanggan harus dapat melakukan perubahan atau menghapus item yang diminta sebelumnya. Tim pengembangan harus menerima persetujuan akhir dari seluruh pemangku kepentingan terkait untuk memastikan hasil akhir memenuhi kebutuhan mereka.
2. Memverifikasi desain: Tim penguji harus meninjau desain perangkat lunak, termasuk tata letak dan prototipe. Apakah model database logis mencakup semua bidang yang relevan untuk menangani data yang lewat bolak-balik antar aplikasi? Apakah pelanggan puas dengan antarmuka yang diusulkan ramah pengguna ? Harus ada konsensus terpadu mengenai validitas dokumen desain sebelum melanjutkan.
3. Memverifikasi logika kode: Tim penguji meninjau kode untuk memastikan kode tersebut lengkap dan konsisten. Mereka juga harus memeriksa kode sumber dan artefak terkait, seperti desain database fisik, dan memastikan semuanya sesuai dengan spesifikasi kebutuhan.

Setelah tim penguji memastikan bahwa semua hal di atas memenuhi standar yang disyaratkan, proses verifikasi dimulai:

1. Perencanaan: Tim penguji memeriksa hubungan antara persyaratan yang berbeda, kemudian menentukan semua tugas yang perlu diselesaikan selama verifikasi dan sesuai jadwal. Sumber daya diberi tanggung jawab untuk tugas-tugas lain, termasuk mendemonstrasikan seberapa baik fungsi berbagai komponen, meninjau jaminan kualitas, atau mengidentifikasi dan membuat katalog potensi cacat.
2. Eksekusi: Sumber daya menjalankan tugas yang diberikan kepada mereka dan mendokumentasikan hasil pengujian verifikasi.
3. Pelaporan: Semua hasil yang dikumpulkan selama pengujian verifikasi dikompilasi menjadi laporan.

Pengujian validasi adalah proses kompleks yang melibatkan pencarian dan pengujian setiap kebutuhan atau

persyaratan pengguna untuk memastikan semuanya berfungsi dengan baik. Mula-mula, tim penguji harus memastikan dahulu bahwa semua hal berikut ini memenuhi standar yang berlaku:

1. Kualifikasi desain: Proses kualifikasi desain, atau DQ, mencakup pembuatan daftar persyaratan bisnis pengguna akhir dan merancang rencana pengujian validasi untuk mengatasinya sebelum meluncurkan produk. Rencana ini juga dapat menjadi catatan tertulis yang berguna mengenai spesifikasi desain yang diinginkan pengembang dan konsumen. Setelah menulis rencana pengujian, tim pengembangan dapat meminta persetujuan dari manajer atau pemegang saham sebelum mereka memulai proses pengujian.
2. Kualifikasi instalasi: Kualifikasi instalasi, atau IQ, melibatkan instalasi perangkat lunak sesuai dengan rencana pengujian validasi. Tim pengembangan produk dapat memastikan bahwa perangkat keras sistem dan proses instalasi itu sendiri sesuai dengan spesifikasi desain. Fase ini juga melibatkan memastikan bahwa lingkungan pengujian cocok untuk pengoperasian produk dan cocok dengan lingkungan di mana produk tersebut kemungkinan besar akan berfungsi setelah perusahaan merilisnya ke publik.
3. Kualifikasi operasional: Kualifikasi operasional, atau OQ, melibatkan pengujian produk dengan berbagai operasi pengujian untuk memastikan produk memenuhi persyaratan pengguna yang ditentukan. Teknik pengujian validasi yang penting mencakup pengujian unit, pengujian integrasi, dan pengujian sistem. Ini semua adalah jenis pengujian fungsionalitas yang berbeda, yang dapat menentukan apakah berbagai elemen perangkat lunak berfungsi sesuai dengan kebutuhan pengguna.

Tim pengembangan perangkat lunak dapat mengikuti rencana dari tahap kualifikasi desain untuk memastikan mereka menguji setiap spesifikasi produk di lingkungan yang sesuai. Hal ini berguna bagi anggota tim pengembangan produk untuk mencatat pengujian ini dengan cermat untuk membuat catatan

tertulis tentang kinerja perangkat lunak. Mereka juga dapat mencatat pembuatan kiriman dan memastikan kiriman tersebut memuaskan melalui proses persetujuan kiriman.

Setelah tim penguji memastikan bahwa semua hal di atas memenuhi standar yang disyaratkan, proses validasi dimulai:

1. Pengujian unit: Pengujian unit adalah bentuk pengujian validasi yang melibatkan penilaian potongan kecil kode secara individual. Unit dapat mencakup potongan kode seperti fungsi, metode, prosedur, modul, atau objek. Menguji unit-unit ini secara terpisah satu sama lain dapat membantu memastikan bahwa masing-masing unit berkinerja baik. Hal ini meningkatkan kemungkinan perangkat lunak berfungsi dengan baik secara keseluruhan. Pengembang juga dapat menggunakan proses pengujian unit untuk membuat catatan unit dan fungsinya, yang mungkin berguna untuk pembaruan perangkat lunak di masa mendatang. Ada beberapa teknik berbeda yang dapat membantu pengembang melakukan pengujian unit. Salah satu teknik pengujian unit dikenal sebagai pengujian kotak hitam. Ini melibatkan pengujian suatu unit tanpa pengetahuan sebelumnya tentang kode atau fungsinya. Ini adalah kebalikan dari pengujian kotak putih, juga dikenal sebagai pengujian kotak kaca atau pengujian kotak bening. Dalam pengujian kotak putih, pengembang menggunakan pengetahuan mereka tentang struktur data internal dan arsitektur perangkat lunak kode sumber untuk menguji fungsionalitas unit. Pengujian kotak hitam dan kotak putih adalah teknik yang dapat digunakan pengembang untuk pengujian unit dan prosedur pengujian validasi lainnya.
2. Pengujian integrasi: Setelah melakukan pengujian unit untuk mengetahui apakah setiap unit kode dalam perangkat lunak berfungsi dengan benar, tim pengembangan perangkat lunak dapat menjalankan pengujian integrasi untuk mempelajari seberapa baik unit berfungsi setelah mereka mengintegrasikannya ke dalam sistem yang lebih besar. Secara khusus, pengembang dapat memastikan bahwa aliran data di seluruh modul berhasil. Dua jenis utama pengujian integrasi adalah pendekatan top-down dan

pendekatan bottom-up. Kedua teknik tersebut melibatkan pengujian integrasi modul perangkat lunak dalam proses langkah demi langkah yang sistematis. Dalam pendekatan top-down, pengembang memulai dengan menghubungkan modul tingkat tinggi dan menelusuri aliran kontrol struktur arsitektur perangkat lunak dari atas ke bawah. Pengujian perangkat lunak dari bawah ke atas melibatkan pengujian elemen tingkat rendah terlebih dahulu dan mengerjakan struktur arsitektur bagian atas.

3. Pengujian sistem Juga dikenal sebagai pengujian tingkat sistem atau pengujian integrasi sistem, jenis pengujian validasi ini dapat menilai perangkat lunak sebagai sistem yang lengkap. Hal ini dapat membantu memastikan bahwa produk berfungsi sesuai dengan spesifikasi sistem end-to-end. Sebagian besar bentuk pengujian sistem melibatkan teknik pengujian validasi kotak hitam. Berikut adalah tiga subkategori spesifik teknik pengujian sistem:
   a. Pengujian asap: Pengujian asap melibatkan pengujian elemen fungsional penting dari produk perangkat lunak. Misalnya, pengujian asap untuk aplikasi perpustakaan musik mungkin melibatkan penilaian kemampuan pengguna untuk masuk, mencari musik, mendengarkan musik, dan menambahkannya ke perpustakaan musik pribadi.
   b. Pengujian kewarasan: Setelah pengujian asap, pengembang dapat membuat perubahan pada kode perangkat lunak, kemudian melakukan pengujian kewarasan untuk menilai fungsionalitas perangkat lunak setelah perubahan kode tersebut.
   c. Pengujian regresi: Pengujian regresi adalah langkah terakhir dalam proses pengujian sistem. Ini melibatkan pencarian dan perbaikan bug baru yang mungkin timbul dari perubahan yang dilakukan pada langkah pengujian sebelumnya untuk memastikan bahwa seluruh sistem perangkat lunak masih berfungsi sebagaimana mestinya.
4. Pengujian kualifikasi kinerja memvalidasi bahwa suatu produk dapat bekerja sesuai dengan kebutuhan bisnis di

dunia nyata. Pengembang di tim internal dapat melakukan pengujian alfa untuk menilai fungsionalitas perangkat lunak dalam simulasi kondisi dunia nyata. Setelah melakukan pengujian mereka sendiri, tim pengembangan produk dapat menawarkan klien eksternal kesempatan untuk menguji produk melalui proses yang disebut pengujian beta. Dalam pengujian beta, klien eksternal dapat menggunakan produk dan mengidentifikasi bug atau tantangan teknis. Bentuk kegiatan yang bisa dilakukan oleh klien eksternal:

a. Tinjauan Sejawat – Ini akan melibatkan tim pembuat perangkat lunak lain dan meminta mereka untuk meninjaunya. Mereka dapat berbagi pandangan mereka tentang kualitas produk dan membantu mengidentifikasi kekurangannya.
b. Inspeksi – Tim tertentu memeriksa produk perangkat lunak dan membantu menemukan potensi kesalahan dan bagian penting.
c. Panduan – Ini melibatkan demonstrasi perangkat lunak kepada sekelompok orang yang bebas menanyakan keraguan mereka. Cara ini dapat membantu menemukan potensi masalah.
d. *Desk-Checking* – Ini adalah proses peninjauan kode sumber perangkat lunak. Tim pengembangan biasanya memeriksa program untuk memastikan algoritme berfungsi dengan baik.

Klien eksternal kemudian dapat melaporkan informasi ini kembali ke tim pengembangan, memberikan kesempatan kepada tim pengembangan untuk melakukan perubahan sebelum merilis produk ke publik. Setelah menerima persetujuan dari penguji beta, produk mungkin sudah siap untuk diproduksi dan diterapkan.

5. Pengujian penerimaan pengguna atau UAT, adalah bentuk pengujian beta kualifikasi kinerja yang mengundang klien untuk menguji produk dan memastikan bahwa produk tersebut memenuhi kebutuhan mereka. Ini sering kali merupakan tahap terakhir dalam proses pengujian validasi dan dapat membantu karena ini menguji produk di lingkungan yang sesuai. Hal ini juga dapat membantu

mengungkap tantangan yang mungkin tidak disadari oleh pengembang karena mereka sudah begitu familiar dengan produk tersebut. Untuk melakukan UAT, pengembang dapat berkolaborasi dengan klien untuk membuat rencana UAT dan mengembangkan skenario pengujian dengan data langsung. Penguji UAT dapat memantau hasil dan membuat log kerusakan untuk mencatat bug atau area tantangan apa pun. Mereka kemudian dapat mengembalikan laporan ini kepada pengembang yang mungkin memperbarui perangkat lunak sebelum merilisnya ke publik.

Setelah menyelesaikan semua tingkat pengujian validasi, produk perangkat lunak dapat diproduksi. Artinya produk siap dipasarkan dan dijual ke konsumen. Tim pengembangan perangkat lunak mungkin membantu memfasilitasi proses penerapan dan instalasi. Perusahaan juga dapat menawarkan bantuan teknis kepada individu yang mengalami kendala teknis ringan pada produknya. Jika mereka menemukan bug teknis yang besar, tim pengembangan dapat mengatasinya dengan merancang solusi dan merilis pembaruan perangkat lunak.

Integrasi otomatisasi dalam pengujian verifikasi dan validasi menawarkan banyak keuntungan. Pengujian otomatis dapat dijalankan secara konsisten dan berulang-ulang, sehingga mengurangi kesalahan manusia dan mempercepat proses pengujian. Hal ini memungkinkan identifikasi cacat secara dini, sehingga menghasilkan perbaikan bug yang lebih cepat dan produk akhir yang lebih kuat. Selain itu, otomatisasi memungkinkan cakupan pengujian yang lebih luas, karena eksekusi kasus pengujian dalam jumlah besar dapat dilakukan dalam rentang waktu yang singkat.

Berikut ini daftar alat otomasi test populer untuk verifikasi dan validasi:

1. Testsigma adalah alat otomatisasi pengujian tanpa kode yang memungkinkan anda mengotomatiskan pengujian dalam bahasa Inggris sederhana, tanpa perlu belajar kode. Bagian terbaiknya adalah ini memungkinkan anda mengotomatiskan pengujian untuk web, seluler, API, dan aplikasi desktop dari tempat yang sama dan itu juga 5x lebih

cepat. Testsigma adalah solusi lengkap berbasis cloud, jadi anda hanya perlu membuat akun di Testsigma, dan anda dapat mulai mengotomatiskan pengujian anda, menjalankannya, membuat laporan untuk pengujian tersebut, dan mengintegrasikannya dengan pipeline CI/CD yang ada.

2. Selenium adalah kerangka otomatisasi sumber terbuka yang banyak digunakan untuk aplikasi web. Ini menyediakan serangkaian alat dan pustaka untuk otomatisasi browser, memungkinkan penguji menulis skrip dalam berbagai bahasa pemrograman seperti Java, Python, atau C#. Selenium dapat mensimulasikan interaksi pengguna, melakukan tindakan seperti mengklik tombol atau memasukkan teks, dan memverifikasi hasil yang diharapkan. Ini biasanya digunakan untuk pengujian fungsional dan pengujian regresi aplikasi web.
3. Appium adalah proyek sumber terbuka dan ekosistem perangkat lunak terkait, yang dirancang untuk memfasilitasi otomatisasi UI pada banyak platform aplikasi, termasuk seluler (iOS, Android, Tizen), browser (Chrome, Firefox, Safari), desktop (macOS, Windows), TV (Roku, tvOS, Android TV, Samsung), dan banyak lagi.

8.6 Kesimpulan

Kualitas produk perangkat lunak yang dibuat sangat ditentukan oleh prosedur yang digunakan untuk membuat dan memeliharanya. Prosedur Verifikasi dan Validasi (V&V) merupakan bagian penting dari siklus hidup perangkat lunak karena V&V menyediakan sarana untuk menilai keberhasilan dan kualitas perangkat lunak. V&V perangkat lunak melaksanakan analisis persyaratan, tinjauan struktur, penelusuran persyaratan, inspeksi kode, tinjauan desain, dan pengujian validasi. Dalam pelaksanaan Verifikasi dan Validasi, penting untuk memastikan bahwa aturan-aturan tertentu dipatuhi pada saat pengembangan produk perangkat lunak, memastikan fakta bahwa produk perangkat lunak yang dikembangkan memenuhi spesifikasi yang dibutuhkan,

memastikan perangkat lunak memenuhi standar kualitas, mengurangi risiko kegagalan atau kesalahan perangkat lunak, meningkatkan efisiensi dan efektivitas proses pengembangan perangkat lunak, membantu menjamin bahwa perangkat lunak memenuhi kriteria pelanggan, peraturan atau industri, mengurangi kemungkinan kegagalan atau cacat perangkat lunak yang dapat membahayakan keselamatan, keamanan atau kinerja, mendeteksi dan memperbaiki masalah lebih awal dalam proses untuk menghindari pengerjaan ulang dan menghemat waktu dan sumber daya, serta menghadirkan perangkat lunak yang memenuhi atau melampaui harapan pengguna sehingga meningkatkan kepuasan dan kepercayaan pengguna.

DAFTAR PUSTAKA

Altaie, A.M., Alsarraj, R.G. and Al-Bayati, A.H. 2020. 'Verification and validation of a software: a review of the literature', *Iraqi Journal for Computers and Informatics*, 46(1), pp. 40–47.

Baduel, R. *et al.* 2018. 'SysML models verification and validation in an industrial context: Challenges and experimentation', in *Modelling Foundations and Applications: 14th European Conference, ECMFA 2018, Held as Part of STAF 2018, Toulouse, France, June 26-28, 2018, Proceedings 14*. Springer, pp. 132–146.

Bondarev, S.E., Chudinov, M.A. and Prokhorov, A.S. 2019. 'The analysis of existing methods of software verification', in *2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*. IEEE, pp. 191–193.

Cheng, S. *et al.* 2019. 'Optimized sampling design and rationale for verification and validation', *Quality and Reliability Engineering International*, 35(1), pp. 483–502.

Delmas, R., Pires, A.F. and Polacsek, T. 2013. 'A verification and validation process for model-driven engineering', *Progress in Flight dynamics, guidance, navigation, control, fault detection, and avionics*, 6, pp. 455–468.

Drusinsky, D., Michael, J.B. and Shing, M.-T. 2008. 'A visual tradeoff space for formal verification and validation techniques', *IEEE Systems Journal*, 2(4), pp. 513–519.

Engels, G. *et al.* 2003. 'Model-based verification and validation of properties', *Electronic Notes in Theoretical Computer Science*, 82(7), pp. 133–150.

Feldt, R. *et al.* 2010. 'Challenges with software verification and validation activities in the space industry', in *2010 third international conference on software testing, verification and validation*. IEEE, pp. 225–234.

Gogolla, M., Hilken, F. and Doan, K.-H. 2018. 'Achieving model quality through model validation, verification and exploration', *Computer Languages, Systems & Structures*,

54, pp. 474–511.
IEEE (2004) *IEEE Std. 1012-2012 IEEE standard for system and software verification and validation*. Available at: https://doi.org/10.1109/IEEESTD.2012.6204026.
Karmakar, G. *et al.* 2023. *Development of Safety-Critical Systems: Architecture and Software*. Springer Nature.
Kezadri, M. and Pantel, M. 2012. 'First steps toward a verification and validation ontology', in *Embedded Real Time Software and Systems (ERTS2012)*.
Mendoza, I. *et al.* 2019. 'Relating verification and validation methods to software product quality characteristics: results of an expert survey', in *Software Quality: The Complexity and Challenges of Software Engineering and Software Quality in the Cloud: 11th International Conference, SWQD 2019, Vienna, Austria, January 15–18, 2019, Proceedings 11*. Springer, pp. 33–44.
Mobin, M. *et al.* 2019. 'An approach for design verification and validation planning and optimization for new product reliability improvement', *Reliability Engineering & System Safety*, 190, p. 106518.
Monteiro, P., Machado, R.J. and Kazman, R. 2009. 'Inception of software validation and verification practices within CMMI Level 2', in *2009 Fourth International Conference on Software Engineering Advances*. IEEE, pp. 536–541.
Nastov, B. *et al.* 2016. 'Towards V&V suitable domain specific modeling languages for MBSE: A tooled approach', in *INCOSE International Symposium*. Wiley Online Library, pp. 556–570.
Nastov, B. *et al.* 2017. 'MBSE and V&V: a tool-equipped method for combining various V&V strategies', *IFAC-PapersOnLine*, 50(1), pp. 10538–10543.
Păsăreanu, C.S. and Bobaru, M. 2012. 'Learning techniques for software verification and validation', in *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Springer, pp. 505–507.

Runeson, P. *et al.* 2006. 'What do we know about defect detection methods?[software testing]', *IEEE software*, 23(3), pp. 82–90.

Schumann, J. and Goseva-Popstojanova, K. 2019. Verification and validation approaches for model-based software engineering', in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, pp. 514–518.

Sen, S., Marijan, D. and Gotlieb, A. 2018. 'Certus: an organizational effort towards research-based innovation in software verification and validation', *International Journal of System Assurance Engineering and Management*, 9, pp. 313–322.

Sherriff, M. and Williams, L. 2006. 'Defect density estimation through verification and validation', in *The 6th Annual High Confidence Software and Systems Conference, Lithicum Heights, MD*, pp. 111–117.

Standard, I. 2008. 'Systems and software engineering–system life cycle processes', *ISO Standard*, 15288, p. 2008.

Walden, D.D. 2015. 'Systems engineering handbook: A guide for system life cycle processes and activities', *(No Title)* [Preprint].

BAB 9
MASA DEPAN REKAYASA KEBUTUHAN DAN PEMODELAN

Oleh Arief Yanto Rukmana

9.1 Pendahuluan

Eksplorasi kebutuhan teknik dan pemodelan masa depan mengungkap lanskap yang ditandai dengan tren transformatif dan lintasan inovatif. Salah satu jalan penting untuk terobosan revolusioner pengembangan di masa depan terletak pada integrasi berkelanjutan dari Kecerdasan Buatan (AI) dan Pembelajaran Mesin (ML) di bidang rekayasa persyaratan (Mayefis *et al.*, 2023). Masa depan menjanjikan alat canggih berbasis AI untuk perolehan persyaratan, analisis, dan pemodelan prediktif. Alat-alat ini diharapkan dapat merevolusi efisiensi dan keakuratan pemahaman kebutuhan pemangku kepentingan, memberikan kerangka kerja yang lebih adaptif dan responsif untuk manajemen persyaratan (R. D. Irawan *et al.*, 2023).

Teknologi Blockchain siap memainkan peran penting dalam membentuk masa depan rekayasa kebutuhan. Karakteristik bawaannya, termasuk desentralisasi dan kekekalan, mengatasi tantangan penting dalam penelusuran dan pengelolaan persyaratan (de Souza, 2022). Munculnya kontrak pintar, perjanjian yang dijalankan sendiri di blockchain, diharapkan merevolusi cara perubahan persyaratan dicatat dan dilaksanakan. Masa depan mungkin akan menyaksikan peningkatan adopsi blockchain untuk memastikan integritas, akuntabilitas, dan keamanan dokumentasi persyaratan (Sari, 2021).

Lintasan arah masa depan dalam rekayasa persyaratan juga mencakup evolusi kolaborasi interdisipliner. Ketika sistem perangkat lunak menjadi semakin kompleks dan saling berhubungan, kolaborasi antara berbagai domain dan

pemangku kepentingan menjadi sangat diperlukan (Coelho *et al.*, 2020). Praktik masa depan dalam rekayasa persyaratan kemungkinan besar akan melibatkan pengembangan metodologi dan alat yang memfasilitasi komunikasi dan kolaborasi efektif antar tim multidisiplin, menjembatani kesenjangan antara spesialis teknis, pengguna akhir, dan pemangku kepentingan lainnya (Arisantoso, Harjanti and Yulianti, 2022).

Munculnya teknologi Edge Computing dan Internet of Things (IoT) memperkenalkan dimensi lain pada masa depan rekayasa kebutuhan. Ketika daya komputasi semakin mendekati sumber pembangkitan data, tantangan unik yang terkait dengan sistem Edge Computing dan IoT harus diatasi (Tokmakov *et al.*, 2019). Praktik masa depan dalam rekayasa persyaratan diharapkan dapat beradaptasi dengan kebutuhan spesifik paradigma komputasi terdistribusi dan terdesentralisasi, memastikan penangkapan persyaratan yang efektif untuk sistem yang beroperasi di tepi jaringan (Baali *et al.*, 2023).

Pertimbangan etis dan penggunaan AI yang bertanggung jawab diharapkan menjadi aspek integral dari masa depan rekayasa persyaratan. Ketika alat berbasis AI menjadi lebih umum, praktik di masa depan kemungkinan besar akan fokus pada penetapan kerangka kerja dan standar untuk mengatur penggunaan AI secara etis dalam rekayasa persyaratan. Mengatasi potensi bias dalam perolehan dan analisis persyaratan akan menjadi hal yang terpenting, sehingga mendorong transparansi dan kepercayaan di antara para pemangku kepentingan (Permana *et al.*, 2023).

Integrasi Augmented Reality (AR) dan Virtual Reality (VR) mewakili aspek transformatif dari lanskap masa depan. Praktik di masa depan mungkin akan menyaksikan adopsi alat AR dan VR secara luas, yang memungkinkan pemangku kepentingan berinteraksi dengan representasi tiga dimensi dari sistem perangkat lunak (Liberatore and Pollack-Johnson, 2003). Pengalaman mendalam ini memiliki potensi untuk meningkatkan pemahaman, kolaborasi, dan pengambilan keputusan dalam proses rekayasa persyaratan (A. S. Y. Irawan *et al.*, 2023).

Masa depan kebutuhan teknik dan pemodelan ditandai dengan konvergensi kemajuan teknologi, pertimbangan etis, dan peningkatan penekanan pada kolaborasi interdisipliner (Triandini *et al.*, 2022). Menavigasi lanskap masa depan ini memerlukan pendekatan berpikiran maju, di mana para praktisi dan peneliti secara aktif terlibat dengan teknologi baru, pertimbangan etis, dan paradigma kolaborasi yang terus berkembang untuk memenuhi tantangan lanskap pengembangan perangkat lunak di masa depan (Rizal *et al.*, 2023).

9.2 Perspektif Sejarah

Perspektif historis mengenai rekayasa persyaratan dan pemodelan sistem perangkat lunak memberikan latar belakang penting untuk memahami evolusi praktik di lapangan. Pada tahap awal pengembangan perangkat lunak, pendekatan tradisional seperti Model Air Terjun mendominasi lanskap. Selama era ini, rekayasa persyaratan terutama berfokus pada pembuatan spesifikasi komprehensif sebelum melanjutkan ke tahap pengembangan berikutnya (Acuña *et al.*, 2012). Namun, metode-metode ini menghadapi keterbatasan dan tantangan yang signifikan, termasuk kurangnya kemampuan beradaptasi terhadap perubahan dinamika proyek dan kesulitan dalam mengakomodasi kebutuhan pemangku kepentingan yang terus berubah (Wahdiniawati *et al.*, 2023).

Munculnya pemodelan sistem perangkat lunak menandai perubahan paradigma yang penting. Bahasa pemodelan, terutama Unified Modeling Language (UML), menjadi terkenal sebagai alat yang ampuh untuk memvisualisasikan dan mendokumentasikan berbagai aspek sistem perangkat lunak. UML, dengan beragam rangkaian diagramnya, menyediakan sarana standar untuk mewakili persyaratan, arsitektur, dan elemen desain. Integrasi pemodelan ke dalam proses rekayasa persyaratan menunjukkan kemajuan yang signifikan, memungkinkan pemangku kepentingan untuk memvisualisasikan komponen dan interaksi sistem dengan lebih intuitif (Bakri *et al.*, 2023).

Narasi sejarah juga menyoroti kelemahan praktik-praktik awal, menekankan perlunya pendekatan yang lebih dinamis dan berulang. Sifat kaku Model Air Terjun dan tantangan yang terkait dengan pembuatan spesifikasi yang lengkap menyebabkan eksplorasi metodologi pengembangan alternatif. Periode ini menyaksikan munculnya model tangkas dan berulang, di mana rekayasa persyaratan menjadi bagian integral dari proses pengembangan yang sedang berlangsung. Pendekatan kolaboratif dan peningkatan keterlibatan pemangku kepentingan menjadi prinsip utama, yang mendorong kemampuan beradaptasi dan tanggap terhadap perubahan kebutuhan proyek (Boje *et al.*, 2020).

Dalam konteks evolusi historis, bab ini menyelidiki interaksi yang rumit antara praktik rekayasa persyaratan dan pematangan pemodelan sistem perangkat lunak. Ini menguraikan bagaimana tantangan awal membuka jalan bagi integrasi bahasa pemodelan seperti UML, memungkinkan representasi persyaratan yang lebih visual dan komunikatif (Arif, Mohammad and Sadiq, 2020). Perspektif historis berfungsi sebagai landasan untuk memahami keadaan rekayasa kebutuhan kontemporer dan menyoroti metodologi berulang dan adaptif yang telah menjadi lazim dalam praktik pengembangan perangkat lunak modern. Bagian ini menyiapkan panggung untuk diskusi selanjutnya mengenai tantangan, peluang, tren yang muncul, dan arah masa depan dalam rekayasa dan pemodelan persyaratan (Pan and Zhang, 2021).

9.3 Keadaan Kebutuhan Teknik Saat Ini

Keadaan rekayasa kebutuhan saat ini mencerminkan perubahan transformatif dalam praktik pengembangan perangkat lunak, yang ditandai dengan penerapan metodologi rekayasa kebutuhan kontemporer. Berbeda dengan pendekatan tradisional, lanskap yang ada dicirikan oleh model pembangunan yang tangkas dan berulang. Metodologi ini memprioritaskan kolaborasi, daya tanggap, dan kemampuan beradaptasi terhadap perubahan dinamika proyek. Keterlibatan pemangku kepentingan ditekankan sepanjang proses

pembangunan, untuk memastikan bahwa kebutuhan yang terus berkembang dapat segera diidentifikasi dan ditangani (Westermann and Evins, 2019).

Dalam kerangka ini, Unified Modeling Language (UML) memainkan peran penting dalam rekayasa persyaratan. UML menawarkan beragam rangkaian diagram yang memfasilitasi visualisasi dan komunikasi kebutuhan sistem yang kompleks. Berbagai diagram UML, seperti diagram use case, diagram aktivitas, dan diagram urutan, digunakan untuk menangkap berbagai aspek persyaratan, memberikan representasi standar dan visual yang meningkatkan pemahaman di antara para pemangku kepentingan (Silva *et al.*, 2019a).

Praktik rekayasa persyaratan kontemporer menekankan sifat dinamis pengembangan perangkat lunak, mendorong kolaborasi berkelanjutan dan putaran umpan balik. Metodologi tangkas, seperti Scrum dan Kanban, memprioritaskan penyampaian peningkatan perangkat lunak secara bertahap, memungkinkan penyempurnaan persyaratan secara terus-menerus berdasarkan umpan balik pengguna. Pendekatan berulang ini mendorong proses pengembangan yang lebih responsif, menyelaraskan sistem perangkat lunak dengan kebutuhan pengguna akhir yang terus berkembang (Triandini *et al.*, 2022).

Keadaan rekayasa kebutuhan saat ini juga ditandai dengan integrasi rekayasa kebutuhan ke dalam konteks DevOps (Pengembangan dan Operasi) yang lebih luas. Kolaborasi yang lancar antara tim pengembangan dan operasi difasilitasi melalui praktik integrasi berkelanjutan dan penerapan berkelanjutan (CI/CD). Integrasi ini memastikan bahwa persyaratan tidak hanya diidentifikasi dan ditentukan secara efektif tetapi juga dialihkan secara mulus ke dalam jalur pengembangan, sehingga menghasilkan rilis perangkat lunak yang lebih cepat dan lebih andal (Arif, Mohammad and Sadiq, 2020).

Meskipun ada kemajuan, tantangan tetap ada dalam kondisi rekayasa kebutuhan saat ini. Menyeimbangkan kebutuhan akan fleksibilitas dengan kebutuhan akan dokumentasi yang komprehensif tetap menjadi pertimbangan yang berkelanjutan. Selain itu, sifat tim pengembangan yang

global dan terdistribusi menimbulkan tantangan dalam komunikasi dan kolaborasi. Seiring dengan terus berkembangnya teknologi, kondisi rekayasa kebutuhan saat ini merupakan lanskap dinamis yang memerlukan pemahaman berbeda mengenai praktik tradisional dan modern agar dapat bernavigasi secara efektif. Bagian ini berfungsi sebagai landasan untuk mengeksplorasi tantangan, peluang, dan tren yang muncul dalam konteks rekayasa kebutuhan dan pemodelan sistem perangkat lunak yang lebih luas (Silva *et al.*, 2019b).

9.4 Tantangan dan Peluang

Gambar 9.1. Tantangan dan Peluang di Masa Depan
(Sumber : https://www.koalahero.com/)

Lanskap rekayasa persyaratan dan pemodelan sistem perangkat lunak penuh dengan banyak sekali tantangan dan peluang, yang mencerminkan sifat rumit pengembangan perangkat lunak modern (Naim *et al.*, 2023). Salah satu tantangan utama terletak pada meningkatnya kompleksitas sistem perangkat lunak dan pesatnya evolusi teknologi (Sutaguna *et al.*, 2023). Ketika sistem menjadi lebih rumit, identifikasi, spesifikasi, dan pengelolaan persyaratan menjadi lebih menantang. Untuk mengatasi kompleksitas ini memerlukan metodologi dan alat canggih yang mampu

menangani saling ketergantungan yang rumit dan lanskap teknologi yang terus berkembang (de Souza, 2022).

Globalisasi dan prevalensi tim pembangunan yang terdistribusi menghadirkan tantangan berat lainnya. Kolaborasi lintas batas geografis menimbulkan hambatan komunikasi dan memerlukan pengembangan strategi untuk memastikan koordinasi dan pemahaman yang efektif di antara beragam pemangku kepentingan. Tantangan ini memerlukan pendekatan inovatif terhadap rekayasa persyaratan kolaboratif, yang menekankan saluran komunikasi yang jelas dan alat yang memfasilitasi kerja tim global yang lancer (Arisantoso, Harjanti and Yulianti, 2022).

Aspek penting dari tantangan dalam rekayasa persyaratan adalah sifat dinamis dari kebutuhan dan harapan pengguna. Dalam lingkungan teknologi yang bergerak cepat, kebutuhan pengguna dapat berkembang dengan cepat, memerlukan pendekatan yang fleksibel dan responsif terhadap rekayasa kebutuhan (Triandini *et al.*, 2022). Tantangannya terletak pada menyeimbangkan kebutuhan akan kelincahan dengan kebutuhan akan persyaratan yang stabil dan terdokumentasi dengan baik, memastikan bahwa perubahan dikelola secara efektif tanpa mengorbankan integritas sistem perangkat lunak (Mayefis *et al.*, 2023).

Kolaborasi antar disiplin ilmu semakin memperumit kondisi ini. Rekayasa persyaratan melibatkan beragam pemangku kepentingan dengan keahlian yang berbeda-beda, mulai dari spesialis teknis hingga pengguna akhir. Menjembatani kesenjangan antara berbagai bidang dan memastikan komunikasi yang efektif antar tim interdisipliner menimbulkan tantangan yang signifikan. Mengembangkan metodologi yang memfasilitasi kolaborasi dan pemahaman yang lancar di antara beragam pemangku kepentingan sangat penting untuk mengatasi tantangan ini (R. D. Irawan *et al.*, 2023).

Di tengah tantangan ini, muncul peluang untuk inovasi dan peningkatan praktik rekayasa persyaratan. Munculnya Model-Driven Engineering (MDE) mewakili peluang yang signifikan. MDE menekankan penggunaan model sebagai artefak kelas satu di seluruh siklus pengembangan perangkat lunak,

mendorong konsistensi dan ketertelusuran mulai dari persyaratan hingga implementasi. Otomatisasi dan pembuatan kode dari model menghadirkan peluang untuk meningkatkan efisiensi dan mengurangi risiko inkonsistensi antara berbagai fase pengembangan (Rukmana and Sudarmanto, 2023).

Peluang menjanjikan lainnya terletak pada integrasi rekayasa persyaratan dengan praktik DevOps. Saluran Continuous Integration and Continuous Deployment (CI/CD) dapat menyederhanakan transisi dari spesifikasi persyaratan ke penerapan, sehingga mendorong proses pengembangan perangkat lunak yang lebih lancar dan otomatis. Sinergi Agile dan DevOps menawarkan potensi untuk menciptakan lingkungan yang lebih responsif dan adaptif untuk mengelola kebutuhan yang terus berkembang (Mardiani, Judijanto and Rukmana, 2023).

Dalam menghadapi tantangan dan memanfaatkan peluang, pertimbangan etis dan hukum dalam rekayasa persyaratan juga menjadi yang terdepan (Sari *et al.*, 2023). Memastikan transparansi, akuntabilitas, dan kepatuhan terhadap peraturan menjadi hal yang terpenting. Interaksi dinamis antara tantangan dan peluang dalam rekayasa persyaratan menggarisbawahi perlunya pendekatan yang berbeda dan adaptif, dengan mengakui sifat teknologi dan kebutuhan pengguna yang terus berkembang (Almahdali *et al.*, 2023). Bagian ini berfungsi sebagai landasan untuk mengeksplorasi tren yang muncul dan arah masa depan dalam bidang rekayasa kebutuhan dan pemodelan sistem perangkat lunak (Khuan, Andriani and Rukmana, 2023).

9.5 Tren yang Muncul dalam Kebutuhan Teknik dan Pemodelan

Eksplorasi tren yang muncul dalam kebutuhan teknik dan pemodelan mengungkap lanskap yang ditandai dengan perubahan transformatif dan kemajuan teknologi. Salah satu tren yang perlu diperhatikan adalah semakin menonjolnya *Model-Driven Engineering* (MDE). Pendekatan ini menekankan peran sentral model sebagai artefak utama sepanjang siklus

pengembangan perangkat lunak. Integrasi MDE ke dalam praktik rekayasa persyaratan menjanjikan peningkatan konsistensi, ketertelusuran, dan efisiensi dengan memanfaatkan model sebagai landasan untuk desain sistem, analisis, dan pembuatan kode. Tujuan utamanya adalah untuk menyederhanakan proses pengembangan dan mendorong sinergi yang lebih besar antara persyaratan dan implementasi (Rukmana, Rahman, *et al.*, 2023).

Kecerdasan Buatan (AI) dan Pembelajaran Mesin (ML) muncul sebagai alat yang ampuh dalam mengatasi tantangan identifikasi dan analisis kebutuhan. Pemunculan kebutuhan yang cerdas, yang didorong oleh algoritma AI, mempunyai potensi untuk meningkatkan akurasi dan efisiensi pengumpulan kebutuhan pemangku kepentingan. Pemodelan prediktif untuk perubahan kebutuhan, didukung oleh ML, memungkinkan adaptasi proaktif terhadap dinamika proyek yang terus berkembang. Kemampuan berbasis AI ini menandakan perubahan paradigma dalam rekayasa persyaratan, memanfaatkan teknologi canggih untuk mengoptimalkan identifikasi, analisis, dan pengelolaan persyaratan sistem (Hartatik *et al.*, 2023).

Teknologi Blockchain memperkenalkan dimensi baru pada ketertelusuran persyaratan. Sebagai buku besar yang terdistribusi dan terdesentralisasi, blockchain memastikan transparansi, kekekalan, dan keamanan dalam pencatatan dan pelacakan persyaratan (Kwak and Stoddard, 2004). Kontrak pintar, perjanjian yang dijalankan sendiri secara otomatis di blockchain, menawarkan potensi untuk merevolusi manajemen persyaratan dengan menyediakan jejak perubahan yang tahan terhadap gangguan dan dapat diaudit. Penerapan blockchain dalam rekayasa persyaratan bertujuan untuk mengatasi tantangan terkait ketertelusuran, akuntabilitas, dan integritas dokumentasi persyaratan (Rukmana, Zebua, *et al.*, 2023).

Munculnya praktik DevOps telah mengubah lanskap pengembangan perangkat lunak, dan integrasinya dengan rekayasa persyaratan merupakan tren yang terus berkembang. Saluran Continuous Integration and Continuous Deployment (CI/CD) memfasilitasi transisi yang mulus dari spesifikasi

persyaratan ke penerapan perangkat lunak. Sinergi Agile dan DevOps memprioritaskan iterasi yang cepat dan rilis yang sering, memastikan bahwa persyaratan yang berkembang dimasukkan secara efisien ke dalam jalur pengembangan. Tren ini menandakan pendekatan holistik terhadap pengembangan perangkat lunak, di mana rekayasa persyaratan menjadi bagian integral dari proses penerapan yang berkelanjutan dan otomatis (Almahdali *et al.*, 2023).

Seiring dengan terus berkembangnya teknologi, peran Augmented Reality (AR) dan Virtual Reality (VR) dalam rekayasa kebutuhan semakin meningkat. Teknologi imersif ini memberikan representasi persyaratan sistem yang interaktif dan tiga dimensi kepada pemangku kepentingan. AR dan VR meningkatkan kolaborasi dan komunikasi dengan memungkinkan pemangku kepentingan memvisualisasikan dan merasakan sistem yang diusulkan dengan cara yang lebih nyata, sehingga memfasilitasi pemahaman yang lebih mendalam tentang persyaratan yang kompleks (Bakri *et al.*, 2023).

Eksplorasi tren yang muncul dalam kebutuhan teknik dan pemodelan menggarisbawahi sifat dinamis dari bidang ini. Dari integrasi teknologi canggih seperti AI dan blockchain hingga pembentukan kembali praktik pengembangan melalui DevOps dan teknologi imersif, tren ini menandakan pergeseran menuju pendekatan yang lebih adaptif, efisien, dan kolaboratif dalam rekayasa persyaratan. Menavigasi tren yang muncul ini memerlukan pola pikir yang berpikiran maju, di mana para praktisi dan peneliti sama-sama menggunakan solusi inovatif untuk mengatasi tantangan yang terus berkembang dalam pemodelan sistem perangkat lunak dan rekayasa persyaratan (Harto, Rukmana, Subekti, *et al.*, 2023).

9.6 Arah Masa Depan

Eksplorasi arah masa depan dalam rekayasa persyaratan dan pemodelan sistem perangkat lunak mengungkap lanskap yang dibentuk oleh perkembangan teknologi dan sifat dinamis pengembangan perangkat lunak. Salah satu hal yang menonjol adalah meningkatnya integrasi Artificial Intelligence (AI) dan

Machine Learning (ML) di lapangan. Masa depan menjanjikan alat berbasis AI yang lebih canggih untuk perolehan kebutuhan, analisis, dan prediksi. Kemajuan ini bertujuan untuk meningkatkan efisiensi dan keakuratan dalam memahami kebutuhan pemangku kepentingan, menawarkan pendekatan yang lebih proaktif dan adaptif terhadap manajemen persyaratan (Harto, Rukmana, Boari, *et al.*, 2023).

Penerapan teknologi Blockchain diperkirakan akan memainkan peran penting di masa depan rekayasa persyaratan. Karakteristik bawaan Blockchain berupa desentralisasi, kekekalan, dan transparansi mengatasi tantangan penting dalam ketertelusuran dan manajemen persyaratan. Kontrak pintar, perjanjian otomatis dan dijalankan sendiri, diharapkan merevolusi cara perubahan persyaratan dicatat dan dilaksanakan. Masa depan mungkin akan melihat peningkatan adopsi blockchain untuk memastikan integritas, akuntabilitas, dan keamanan dokumentasi persyaratan (Al Aidhi *et al.*, 2023).

Lintasan Arah Masa Depan juga mencakup evolusi kolaborasi interdisipliner. Ketika sistem perangkat lunak menjadi lebih kompleks dan saling berhubungan, kolaborasi antara berbagai domain dan pemangku kepentingan menjadi suatu keharusan (Engemann and O'Connor, 2021). Arah masa depan dalam rekayasa persyaratan melibatkan pengembangan metodologi dan alat yang memfasilitasi komunikasi dan kolaborasi efektif antar tim multidisiplin, menjembatani kesenjangan antara spesialis teknis, pengguna akhir, dan pemangku kepentingan lainnya (Rukmana, Priyana, *et al.*, 2023).

Munculnya teknologi *Edge Computing* dan *Internet of Things* (IoT) menghadirkan arah masa depan lainnya dalam rekayasa persyaratan. Ketika daya komputasi semakin mendekati sumber pembuatan data, persyaratan untuk sistem yang memanfaatkan Edge Computing dan IoT menjadi unik. Praktik masa depan dalam rekayasa persyaratan mungkin perlu mengatasi tantangan spesifik yang terkait dengan paradigma komputasi terdistribusi dan terdesentralisasi, memastikan bahwa persyaratan ditangkap secara efektif untuk sistem yang beroperasi di tepi jaringan (Harto *et al.*, 2022).

Pertimbangan etis dan penggunaan AI yang bertanggung jawab diharapkan menjadi aspek integral dari arah masa depan dalam rekayasa persyaratan. Ketika alat berbasis AI menjadi lebih umum, memastikan pedoman etika dan mengatasi potensi bias dalam perolehan dan analisis persyaratan akan menjadi hal yang sangat penting. Praktik di masa depan kemungkinan besar akan fokus pada penetapan kerangka kerja dan standar untuk mengatur penggunaan AI yang etis dalam rekayasa persyaratan, mendorong transparansi dan kepercayaan di antara para pemangku kepentingan (Razali *et al.*, 2023).

Integrasi *Augmented Reality* (AR) dan *Virtual Reality* (VR) siap merevolusi visualisasi dan komunikasi kebutuhan. Praktik di masa depan mungkin melibatkan adopsi alat AR dan VR secara luas, yang memungkinkan pemangku kepentingan berinteraksi dengan representasi tiga dimensi dari sistem perangkat lunak. Pengalaman mendalam ini dapat meningkatkan pemahaman, kolaborasi, dan pengambilan keputusan dalam proses rekayasa persyaratan (Mustafa *et al.*, 2023).

Arah masa depan dalam rekayasa persyaratan dan pemodelan sistem perangkat lunak mengarah pada lanskap yang dibentuk oleh kemajuan teknologi, pertimbangan etis, dan penekanan berkelanjutan pada kolaborasi interdisipliner. Menavigasi lintasan masa depan ini memerlukan pendekatan berpikiran maju, di mana para praktisi dan peneliti secara aktif terlibat dengan teknologi baru, pertimbangan etis, dan paradigma kolaborasi yang terus berkembang untuk memenuhi tantangan lanskap pengembangan perangkat lunak di masa depan (Rukmana, Meltareza, *et al.*, 2023).

9.6.1 AI dan Pembelajaran Mesin dalam Identifikasi dan Analisis Kebutuhan

Integrasi *Artificial Intelligence* (AI) dan *Machine Learning* (ML) dalam identifikasi dan analisis kebutuhan mewakili perubahan paradigma di bidang rekayasa kebutuhan (Arif, Mohammad and Sadiq, 2020). Teknologi AI dan ML menghadirkan kemampuan canggih yang berpotensi meningkatkan efisiensi, akurasi, dan kemampuan beradaptasi

secara signifikan dari seluruh proses perolehan persyaratan (Wu *et al.*, 2016).

Salah satu penerapan utama AI dalam identifikasi kebutuhan adalah dalam bidang perolehan kebutuhan yang cerdas. Metode tradisional seringkali mengandalkan proses manual dan interaksi langsung dengan pemangku kepentingan untuk memahami kebutuhan. Namun, sistem yang digerakkan oleh AI dapat secara mandiri menganalisis kumpulan data yang sangat besar, interaksi pengguna, dan data proyek historis untuk mengidentifikasi pola dan tren. Hal ini memungkinkan sistem untuk secara proaktif menyarankan persyaratan potensial atau area yang mungkin memerlukan klarifikasi lebih lanjut, sehingga menyederhanakan tahap awal pengumpulan persyaratan (Wagner, 2013).

Machine Learning semakin meningkatkan analisis kebutuhan dengan memungkinkan sistem belajar dari data historis dan hasil proyek. Algoritme ML dapat menganalisis proyek sebelumnya, mengidentifikasi pola keberhasilan, dan memprediksi potensi tantangan berdasarkan konteks serupa. Analisis prediktif ini membantu mengantisipasi potensi modifikasi persyaratan, mengurangi kemungkinan kebutuhan terabaikan, dan berkontribusi pada spesifikasi persyaratan yang lebih kuat dan adaptif (Sudirjo *et al.*, 2023).

Natural Language Processing (NLP), salah satu subbidang AI, sangat berpengaruh dalam identifikasi dan analisis kebutuhan. NLP memungkinkan sistem untuk memahami dan menafsirkan bahasa manusia, memungkinkan perolehan persyaratan yang lebih bernuansa dan sadar konteks (Mens *et al.*, 2005). Alat berbasis AI yang menggunakan NLP dapat mengekstrak wawasan berharga dari data tekstual tidak terstruktur, seperti masukan pengguna, dokumentasi, dan saluran komunikasi (Amato *et al.*, 2019). Hal ini memfasilitasi pemahaman yang lebih komprehensif dan akurat mengenai kebutuhan pemangku kepentingan, bahkan dalam situasi di mana persyaratan mungkin tidak disebutkan secara eksplisit (Hariyati *et al.*, 2023).

Selain itu, AI dan ML berkontribusi terhadap analisis kebutuhan dinamis dengan beradaptasi terhadap perubahan

dinamika proyek. Seiring dengan berkembangnya kebutuhan pemangku kepentingan atau tersedianya informasi baru, sistem AI dapat terus menganalisis dan memperbarui lanskap kebutuhan. Kemampuan beradaptasi secara real-time ini merupakan perubahan besar dari pendekatan statis tradisional, yang memastikan bahwa persyaratan tetap selaras dengan sifat proyek yang terus berkembang (Dwijayani *et al.*, 2023).

Meskipun integrasi AI dan ML dalam identifikasi dan analisis kebutuhan sangat menjanjikan, hal ini juga membawa tantangan seperti perlunya tata kelola data yang kuat, mengatasi bias dalam data pelatihan, dan memastikan transparansi dalam pengambilan keputusan berbasis AI. Pertimbangan etis dalam penggunaan AI, termasuk masalah privasi dan praktik AI yang bertanggung jawab, menjadi aspek krusial yang perlu mendapat perhatian cermat (Chakroborti *et al.*, 2023).

Penggunaan AI dan ML dalam identifikasi dan analisis kebutuhan merevolusi proses rekayasa persyaratan. Teknologi ini memberdayakan sistem untuk mengidentifikasi pola secara mandiri, memprediksi kebutuhan potensial, dan secara dinamis beradaptasi terhadap perubahan dinamika proyek. Penerapan NLP semakin menyempurnakan pemahaman akan kebutuhan pemangku kepentingan, mengantarkan era baru efisiensi dan kemampuan beradaptasi dalam domain rekayasa kebutuhan (Romano, Chen and Nunamaker, 2002).

9.6.2 Teknologi Blockchain dalam Ketertelusuran Persyaratan

Teknologi *Blockchain* muncul sebagai kekuatan transformatif dalam domain ketertelusuran persyaratan dalam rekayasa perangkat lunak. Secara tradisional, ketertelusuran persyaratan melibatkan pembentukan dan pemeliharaan hubungan antara berbagai fase siklus hidup pengembangan perangkat lunak untuk memastikan bahwa setiap persyaratan diterapkan dengan benar dan perubahan dilacak dengan cermat. Blockchain memperkenalkan buku besar yang terdesentralisasi, transparan, dan tahan terhadap gangguan, memberikan solusi unik terhadap tantangan dalam ketertelusuran persyaratan (Wakil *et al.*, 2022).

Salah satu aspek mendasar dari dampak blockchain terhadap ketertelusuran persyaratan adalah sifatnya yang terdesentralisasi. Dalam jaringan blockchain, setiap peserta (node) menyimpan salinan seluruh buku besar. Arsitektur terdesentralisasi ini memastikan bahwa tidak ada satu titik kegagalan pun, dan semua peserta memiliki akses terhadap informasi terkini yang sama. Dalam konteks ketertelusuran persyaratan, ini berarti bahwa pemangku kepentingan di berbagai tim, departemen, atau bahkan organisasi dapat berbagi catatan persyaratan dan evolusinya yang sama dan tersinkronisasi (R. D. Irawan *et al.*, 2023).

Kekekalan adalah fitur utama dari blockchain, sehingga sangat cocok untuk mencatat dan melacak perubahan persyaratan. Setelah satu blok data ditambahkan ke blockchain, blok tersebut tidak dapat diubah secara surut. Kekekalan ini memastikan integritas catatan ketertelusuran persyaratan, mencegah perubahan yang tidak sah atau tidak disengaja. Setiap perubahan atau penambahan pada data ketertelusuran persyaratan diberi stempel waktu secara aman dan ditautkan ke versi sebelumnya, sehingga memberikan riwayat perubahan yang dapat diaudit dan transparan (Karim *et al.*, 2020).

Kontrak cerdas, perjanjian yang dijalankan sendiri dan dapat diprogram pada blockchain, semakin meningkatkan ketertelusuran persyaratan. Kontrak pintar dapat digunakan untuk mengotomatiskan pelaksanaan tindakan yang telah ditentukan sebelumnya berdasarkan kondisi atau peristiwa tertentu dalam proses ketertelusuran persyaratan (Sagiroglu and Sinanc, 2013). Misalnya, kontrak pintar dapat secara otomatis memperbarui catatan ketertelusuran ketika persyaratan baru ditambahkan atau ketika perubahan disetujui. Otomatisasi ini mengurangi kemungkinan kesalahan manusia dan menyederhanakan proses penelusuran (Setiawan *et al.*, 2023).

Keamanan dan kontrol akses sangat penting dalam ketertelusuran persyaratan, terutama di industri yang melibatkan informasi sensitif (Herman, Umar and Marsaid, 2023). Prinsip kriptografi Blockchain memastikan bahwa data yang disimpan di blockchain aman dan anti kerusakan. Kontrol

akses dapat diterapkan untuk menentukan siapa yang dapat melihat atau mengubah catatan ketertelusuran persyaratan tertentu (Munawar and Putri, 2020). Tingkat keamanan ini sangat penting untuk memastikan bahwa hanya individu yang berwenang yang memiliki akses terhadap informasi sensitif terkait persyaratan dan kemampuan penelusurannya (Puspita *et al.*, 2022).

Dalam lingkungan pengembangan yang kolaboratif dan terdistribusi, di mana banyak pemangku kepentingan berkontribusi dan bergantung pada persyaratan bersama, teknologi blockchain memfasilitasi proses penelusuran yang transparan dan akuntabel. Sifat blockchain yang terdesentralisasi dan aman memastikan bahwa semua pemangku kepentingan mempunyai akses terhadap catatan persyaratan yang sama dan tidak dapat diubah, menumbuhkan kepercayaan dan kolaborasi di antara tim yang berbeda (Rachmadi and Kom, 2020).

Meskipun penerapan blockchain dalam ketertelusuran persyaratan membawa banyak manfaat, penting untuk mempertimbangkan tantangan seperti skalabilitas, integrasi dengan sistem yang ada (McAfee *et al.*, 2012), dan dampak lingkungan yang terkait dengan implementasi blockchain tertentu. Seiring dengan semakin matangnya teknologi (Rukmana, 2023), dan dengan kemajuan yang berkelanjutan, blockchain memberikan harapan besar dalam merevolusi cara pengelolaan ketertelusuran persyaratan, memberikan landasan yang kuat, aman, dan transparan untuk bidang rekayasa perangkat lunak yang terus berkembang (Candra, 2019).

DAFTAR PUSTAKA

Acuña, S.T. *et al.* (2012) 'A systematic mapping study on the open source software development process', in *16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012)*. IET, pp. 42–46.

Al Aidhi, A. *et al.* (2023) 'Peningkatan Daya Saing Ekonomi melalui peranan Inovasi', *Jurnal Multidisiplin West Science*, 2(02), pp. 118–134.

Almahdali, H. *et al.* (2023) *NEW TECHNOLOGIES IN TEACHING AND LEARNING*. Get Press Indonesia.

Amato, F. *et al.* (2019) 'Analyse digital forensic evidences through a semantic-based methodology and NLP techniques', *Future Generation Computer Systems*, 98, pp. 297–307.

Arif, M., Mohammad, C.W. and Sadiq, M. (2020) 'Software requirements modeling: a systematic literature review', in *2020 IEEE International Conference on Computing, Power and Communication Technologies (GUCON)*. IEEE, pp. 194–200.

Arisantoso, A., Harjanti, T.W. and Yulianti, S.D. (2022) 'Modul Pembelajaran Rekayasa Perangkat Lunak'.

Baali, Y. *et al.* (2023) *SISTEM INFORMASI MANAJEMEN: KONSEP DAN APLIKASI BISNIS*. Get Press Indonesia.

Bakri, A.A. *et al.* (2023) 'Blockchain Technology and its Disruptive Potential in the Digital Economy', *West Science Journal Economic and Entrepreneurship*, 1(03), pp. 116–123.

Boje, C. *et al.* (2020) 'Towards a semantic Construction Digital Twin: Directions for future research', *Automation in construction*, 114, p. 103179.

Candra, H.K. (2019) *Pengantar Teknologi Informasi*. Poliban Press.

Chakroborti, D. *et al.* (2023) 'Release conventions of open-source software: An exploratory study', *Journal of Software: Evolution and Process*, 35(1), p. e2499.

Coelho, J. *et al.* (2020) 'Is this GitHub project maintained? Measuring the level of maintenance activity of open-source projects', *Information and Software Technology*, 122, p. 106274.

Dwijayani, H. *et al.* (2023) 'Determinant Factors E-Satisfaction and Repurchase Intention of Investment Platform Users In Indonesia', *Jurnal Sistim Informasi dan Teknologi*, pp. 46–51.

Engemann, K.J. and O'Connor, R. V (2021) *Project Risk Management: Managing Software Development Risk*. Walter de Gruyter GmbH & Co KG.

Hariyati, H. *et al.* (2023) 'Management accounting information system and intellectual capital: a way to increase SME's business performance', *Jurnal Siasat Bisnis*, pp. 61–75.

Hartatik, H. *et al.* (2023) *TREN TECHNOPRENEURSHIP: Strategi & Inovasi Pengembangan Bisnis Kekinian dengan Teknologi Digital*. PT. Sonpedia Publishing Indonesia.

Harto, B. *et al.* (2022) 'Bibliometric Analysis of Transforming Leadership Education with Artificial Intelligence', in *1st Virtual Workshop on Writing Scientific Article for International Publication Indexed SCOPUS*. Sciendo, pp. 385–390. Available at: https://doi.org/10.2478/9788366675827-067.

Harto, B., Rukmana, A.Y., Subekti, R., *et al.* (2023) *TRANSFORMASI BISNIS DI ERA DIGITAL: Teknologi Informasi dalam Mendukung Transformasi Bisnis di Era Digital*. PT. Sonpedia Publishing Indonesia.

Harto, B., Rukmana, A.Y., Boari, Y., *et al.* (2023) *WIRAUSAHA BIDANG TEKNOLOGI INFORMASI: Peluang usaha dalam meyongsong era society 5.0*. PT. Sonpedia Publishing Indonesia.

Herman, H., Umar, R. and Marsaid, A.P. (2023) 'Analisis Keamanan Jaringan LAN Terhadap Kerentanan Jaringan Ancaman DDoS Menggunakan Metode Penetration', *JURIKOM (Jurnal Riset Komputer)*, 10(1), pp. 317–329.

Irawan, A.S.Y. *et al.* (2023) *PENGENALAN JARINGAN KOMPUTER*. Get Press Indonesia.

Irawan, R.D. *et al.* (2023) 'Pemodelan Hasil Rekayasa Kebutuhan Perangkat Lunak Sistem Jurnal Elektronik Terintegrasi" Ideogram"', *JIKO (Jurnal Informatika dan Komputer)*, 7(1), pp. 13–22.

Karim, A. *et al.* (2020) *Pengantar teknologi informasi*. Yayasan Labuhanbatu Berbagi Gemilang.

Khuan, H., Andriani, E. and Rukmana, A.Y. (2023) 'The Role of Technology in Fostering Innovation and Growth in Start-up Businesses', *West Science Journal Economic and Entrepreneurship*, 1(03), pp. 124–133.

Kwak, Y.H. and Stoddard, J. (2004) 'Project risk management: lessons learned from software development environment', *Technovation*, 24(11), pp. 915–920.

Liberatore, M.J. and Pollack-Johnson, B. (2003) 'Factors influencing the usage and selection of project management software', *IEEE transactions on Engineering Management*, 50(2), pp. 164–174.

Mansyur, A.R. (2020) 'Dampak COVID-19 Terhadap Dinamika Pembelajaran Di Indonesia', *Education and Learning Journal*, Vol. 1, No, pp. 113–123.

Mardiani, E., Judijanto, L. and Rukmana, A.Y. (2023) 'Improving Trust and Accountability in AI Systems through Technological Era Advancement for Decision Support in Indonesian Manufacturing Companies', *West Science Interdisciplinary Studies*, 1(10), pp. 1019–1027.

Mayefis, R. *et al.* (2023) *Rekayasa Perangkat Lunak*. Global Eksekutif Teknologi.

McAfee, A. *et al.* (2012) 'Big data: the management revolution', *Harvard business review*, 90(10), pp. 60–68.

Mens, T. *et al.* (2005) 'Challenges in software evolution', in *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*. IEEE, pp. 13–22.

Munawar, Z. and Putri, N.I. (2020) 'Keamanan Jaringan Komputer Pada Era Big Data', *J-SIKA| Jurnal Sistem Informasi Karya Anak Bangsa*, 2(01), pp. 14–20.

Mustafa, F. *et al.* (2023) 'The Role of E-Commerce Use, Capital Availability and Business Training on Performance of Small Medium Enterprise (SMEs) in Indonesia', *MALCOM:*

Indonesian Journal of Machine Learning and Computer Science, 3(2).

Naim, S. *et al.* (2023) 'TRANSFORMASI TOKO TRADISIONAL MENJADI TOKO BERBASIS DIGITAL: PELATIHAN DAN PENDAMPINGAN UNTUK MENGHADAPI ERA DIGITALISASI', *Eastasouth Journal of Impactive Community Services*, 1(03), pp. 215–222.

Pan, Y. and Zhang, L. (2021) 'Roles of artificial intelligence in construction engineering and management: A critical review and future trends', *Automation in Construction*, 122, p. 103517.

Permana, A.A. *et al.* (2023) *ARTIFICIAL INTELLIGENCE MARKETING*. Get Press Indonesia.

Puspita, H. *et al.* (2022) *Pengantar Teknologi Informasi*. Haura Utama.

Rachmadi, T. and Kom, S. (2020) *Pengantar Teknologi Informasi*. Tiga Ebook.

Razali, G. *et al.* (2023) 'Loyalty and the Effects of Trust and Switching Barriers', *Jurnal Mirai Management*, 8(1), pp. 237–248.

Rizal, M. *et al.* (2023) *TRANSFORMASI DIGITAL: MEMAHAMI INTERNET OF THINGS*. Get Press Indonesia.

Romano, N.C., Chen, F. and Nunamaker, J.F. (2002) 'Collaborative project management software', in *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*. IEEE, pp. 233–242.

Rukmana, A.Y. (2023) 'BAB 3 TEKNOLOGI DIGITAL', *Digital Marketing Dan E-Commerce*, p. 27.

Rukmana, A.Y., Priyana, Y., *et al.* (2023) 'Dampak Kebijakan Pemerintah Terhadap Ekosistem Kewirausahaan: Studi Kasus Inkubator Bisnis di Indonesia', *Jurnal Ekonomi dan Kewirausahaan West Science*, 1(03), pp. 216–225.

Rukmana, A.Y., Zebua, R.S.Y., *et al.* (2023) *DUNIA MULTIMEDIA: Pengenalan dan Penerapannya*. PT. Sonpedia Publishing Indonesia.

Rukmana, A.Y., Meltareza, R., *et al.* (2023) 'Optimizing the Role of Business Incubators in Higher Education: A Review of

Supporting Factors and Barriers', *West Science Business and Management*, 1(03), pp. 169–175.
Rukmana, A.Y., Rahman, R., *et al.* (2023) *PENGANTAR SISTEM INFORMASI: Panduan Praktis Pengenalan Sistem Informasi & Penerapannya*. PT. Sonpedia Publishing Indonesia.
Rukmana, A.Y. and Sudarmanto, E. (2023) 'Transformasi Bisnis dan Manajemen: Dampak Implementasi Teknologi 5G di Era Konektivitas Cepat', *Jurnal Bisnis dan Manajemen West Science*, 2(03), pp. 226–238.
Sagiroglu, S. and Sinanc, D. (2013) 'Big data: A review', in *2013 international conference on collaboration technologies and systems (CTS)*. IEEE, pp. 42–47.
Sari, I.P. (2021) *Buku Ajar Rekayasa Perangkat Lunak*. umsu press.
Sari, O.H. *et al.* (2023) *DIGITAL MARKETING: Optimalisasi Strategi Pemasaran Digital*. PT. Sonpedia Publishing Indonesia.
Setiawan, Z. *et al.* (2023) *Kewirausahaan Digital*. Global Eksekutif Teknologi.
Silva, J.M. *et al.* (2019a) 'Requirements engineering at a glance: comparing gore and UML methods in the design of automated systems', in *Congresso Brasileiro de Automática-CBA*.
Silva, J.M. *et al.* (2019b) 'Requirements engineering at a glance: comparing gore and UML methods in the design of automated systems', in *Congresso Brasileiro de Automática-CBA*.
de Souza, B.W.L. (2022) 'The era of Social Machines in the BPM-SOA and UML-SOA convergence', *Research, Society and Development*, 11(2), pp. e10011225178–e10011225178.
Sudirjo, F. *et al.* (2023) 'THE EFFECT OF BRAND AWARENESS AND ELECTRONIC SERVICE QUALITY ON LOYALTY OF E-COMMERCE CUSTOMERS', *Jurnal Scientia*, 12(03), pp. 3984–3989.
Sutaguna, I.N.T. *et al.* (2023) 'Training on Business Planning Using Business Canvas Models', *Cakrawala: Jurnal Pengabdian Masyarakat Global*, 2(3), pp. 34–43.

Tokmakov, K. *et al.* (2019) 'A case for data centre traffic management on software programmable ethernet switches', in *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*. IEEE, pp. 1–6.

Triandini, E. *et al.* (2022) 'Software similarity measurements using UML diagrams: A systematic literature review', *Register: Jurnal Ilmiah Teknologi Sistem Informasi*, 8(1).

Wagner, S. (2013) 'Software product quality control'.

Wahdiniawati, S.A. *et al.* (2023) *ENTERPRISE INFORMATION SYSTEM*. Get Press Indonesia.

Wakil, A. *et al.* (2022) *Transformasi Digital Dalam Dunia Bisnis*. Global Eksekutif Teknologi.

Westermann, P. and Evins, R. (2019) 'Surrogate modelling for sustainable building design–A review', *Energy and Buildings*, 198, pp. 170–186.

Wu, H. *et al.* (2016) 'Maintenance effort estimation for open source software: A systematic literature review', in *2016 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, pp. 32–43.

BIODATA PENULIS

Ade Maulana, S.Kom., M.T.I.
Dosen Program Studi Sistem Informasi
Fakultas Ilmu Komputer Universitas Pelita Harapan
Kampus Kota Medan

Ade Maulana lahir di Kota Medan. Pria berdarah minang ini, merupakan lulusan Magister Teknologi Informasi dari Universitas Indonesia. Sebelumnya, Ade bekerja di beberapa perusahaan di bidang teknologi informasi. Namun, pada tahun 2020, ia memutuskan untuk menyisihkan sebagian waktunya untuk mengajar pada program studi Sistem Informasi di Universitas Pelita Harapan, kampus Kota Medan. Ia memiliki minat khusus dalam Software Engineering, Product Management, Blockchain, Green Economy, dan Digital Economy. Selain itu, Ade telah menulis beberapa buku dan jurnal ilmiah yang berkontribusi pada pengetahuan di bidangnya. Ia juga memiliki beberapa sertifikasi keahlian, termasuk Certified in IT Service Management (CITSM) dari Quint Wellington, Android Developer Associate (ADA) dari Logical Operation, IT Specialist : Computational Thinking dari Pearson dan Project Management Ready (PMR) dari PMI.

BIODATA PENULIS

Dr. Irmawati, S.Kom., MMSI.
Dosen Program Studi Sistem Informasi
Fakultas Teknik dan Informatika
Universitas Multimedia Nusantara

Penulis adalah dosen tetap pada Program Studi Sistem Informasi Fakultas Teknik dan Informatika Universitas Multimedia Nusantara. Menyelesaikan pendidikan S1 dan S2 pada Program Studi Sistem Informasi Universitas Gunadarma dan S3 pada departemen Teknik Elektro Universitas Indonesia. Penulis menekuni bidang keilmuan meliputi *Artificial Intelligence*, *Image Processing*, *Machine Learning*, dan *Deep Learning*.

BIODATA PENULIS

Nanang Durahman, ST., M.Kom
Dosen Program Studi Manajemen Informatika

Penulis merupakan satu dosen tetap pada Program Studi Manajemen Informatika STMIK DCI, Ketertarikannya terhadap komputer sejak tahun 2008 silam sejak menduduki Sekolah Menengah Pertama. Hal tersebut membuat penulis memilih untuk masuk ke Sekolah Menengah Kejuruan di SMK DCI dengan memilih Jurusan Teknik Komputer Jaringan dan berhasil lulus pada tahun 2011. melanjutkan pendidikan ke jenjang studi S1 di prodi Teknik Informatika STMIK DCI pada tahun 2015. 2 tahun kemudian, penulis menyelesaikan pendidikan S2 di prodi Sistem Informasi STMIK LIKMI Bandung. Penulis memiliki kepakaran pengajaran dibidang Sistem Multimedia, Sistem Informasi dan Rekeyasa Perangkat Lunak. Selain itu Dan untuk mewujudkan karir sebagai dosen penulis aktif sebagai dalam pelaksanaan penelitian dan pengabdian, beberapa penelitian didanai oleh Kemdikbud. Karyanya sudah dipublikasikan dalam karya ilmiah pada jurnal, prosiding, buku dan HKI. Adapun tulisan artikel lainya penulis mempublikasiakn halamn blog pribadi pada URL http://nanangdurahman.my.id

BIODATA PENULIS

Wirawan Istiono, S.Kom., M.Kom.
Dosen Informatika
Fakultas Teknik dan Informatika, Universitas Multimedia Nusantara

Penulis lahir di Jakarta tanggal 13 April 1983. Penulis adalah dosen pada Program Studi Informatika Fakultas Teknik dan Informatika, Universitas Multimedia Nusantara. Menyelesaikan pendidikan S1 pada Jurusan Teknik Informatika dan melanjutkan S2 pada bidang jurusan yang Teknik Informatika. Pada saat ini, penulis menekuni bidang Penelitian terkait komputer sains, sistem komputer dan teknologi informasi

BIODATA PENULIS

Ir. Johni S. Pasaribu, MT.
Dosen Program Studi Sistem Informasi
Fakultas IT Politeknik Piksi Ganesha

Penulis lahir di Medan tanggal 1 Juni 1966 dari pasangan HA Pasaribu (alm) dan Taruli Tambunan (alm). Penulis adalah anak keempat dari enam bersaudara. Pendidikan yang ditempuh yaitu SD Methodist Medan lulus tahun 1979, SMPN 88 Jakarta lulus tahun 1982 dan SMAN 3 Jakarta lulus tahun 1985. Kemudian melanjutkan studi di ITB jurusan Teknik Fisika spesialisasi Instrumentasi dan Kontrol. Penulis melakukan penelitian untuk tugas akhir di PT IPTN dengan judul Perancangan Sistem Kendali Fly By Wire Gerak Longitudinal Pada Engineering Flight Simulator Pesawat N250-100 dan menyelesaikan gelar sarjana teknik Insinyur (Ir) tahun 1992.

Kemudian penulis bekerja di PT. IPTN (sekarang PT Dirgantara Indonesia) di Departemen Iron Bird kemudian berpindah di Departemen Engineering Flight Simulator (EFS). Penulis terlibat dalam Operation and Maintenance untuk EFS sehingga simulator N250-100 ini dapat digunakan saat dilakukan pengujian-pengujian pesawat melalui simulator sebelum pengujian langsung pesawat di udara. Tahun 1998 penulis melanjutkan studi S2 di ITB pada jurusan Teknik Informatika untuk spesialisasi Software Engineering, dimana ilmu yang diperoleh dapat diaplikasikan dalam pekerjaan di lapangan. Tahun 2001 penulis menyelesaikan pendidikan S2

dengan gelar MT mengambil penelitian yang berjudul Pengembangan Perangkat Lunak Simulasi Sistem Perasa Kendali Elevator Pesawat N250-100 dimana penelitian ini juga dilakukan di Engineering Flight Simulator (EFS) N250-100. Tahun 2004 penulis mulai aktif mengajar di Perguruan Tinggi di Bandung diantaranya pernah mengajar di Unikom, ITHB, STT Telkom (kini TelU). Kini penulis penuh mengajar dan menjadi dosen tetap di Politeknik Piksi Ganesha. Bidang keahlian mengajar dan penelitian penulis adalah Artificial Intelligence (AI), Decision Support System (DSS), Software Engineering dan Software Testing. Penulis juga aktif melakukan bimbingan mahasiswa S1 dan publikasi di jurnal-jurnal nasional maupun internasional.

BIODATA PENULIS

Alexander Waworuntu, S.Kom., M.T.I.
Dosen Informatika, Fakultas Teknik dan Informatika
Universitas Multimedia Nusantara

Penulis lahir di Surabaya pada tanggal 9 Juni. Penulis adalah dosen pada Program Studi Informatika, Fakultas Teknik dan Informatika, Universitas Multimedia Nusantara. Menyelesaikan pendidikan S1 pada jurusan Teknik Informatika pada tahun 2006 di Universitas Bina Nusantara. Selanjutnya penulis menyelesaikan pendidikan S2 pada program studi Magister Teknologi Informasi di Universitas Indonesia pada tahun 2012.

BIODATA PENULIS

Green Ferry Mandias
Dosen Program Studi Informatika
Fakultas Ilmu Komputer
Universitas Klabat

Lahir di Kawangkoan 4 Februari 1981. Menjadi anggota Aptikom pada tahun 2014 sampai saat ini. Menyelesaikan sarjana komputer di Universitas Klabat pada tahun 2003 dan Master of Computer Science di Universitas Gadjah Mada 2011. Aktif sebagai dosen di Fakutas Ilmu Komputer Universitas Klabat dan menjadi Ketua Program Studi Informatika Universitas Klabat pada tahun 2018 – saat ini. Bidang kajian yang diminati ialah kajian data mining, database dan datawarehouse.

Selain aktif sebagai dosen, aktif pula dalam penelitian dan pengembangan di bidang informatika dan sistem informasi. Telah mempublikasikan banyak makalah ilmiah di jurnal nasional dan konferensi international. Sering diundang untuk menjadi pembicara dalam seminar dibeberapa daerah dan mempunyai komitmen untuk memberikan pendidikan yang berkualitas dan menginspirasi mahasiswa menjadi ahli dibidangnya.

BIODATA PENULIS

Raymond Sunardi Oetama
Dosen Sistem Informasi
Universitas Multimedia Nusantara

Raymond Sunardi Oetama adalah seorang dosen full time di bidang Computer Science pada Universitas Multimedia Nusantara sejak tahun 2011, fokus penelitian pada information system, machine learning serta statistics. Raymond menamatkan studi S2 nya bidang Master Computer and Information Science di Auckland University of Technology, New Zealand. Saat ini Raymond sedang melakukan penelitian di bidang deep learning.

BIODATA PENULIS

Arief Yanto Rukmana, S.T., M.M., C.Ed., CBPA., CLMA®
Mahasiswa Doktoral Universitas Pendidikan Indonesia
Dosen Perguruan Tinggi Indonesia Mandiri Bandung
Dosen Universitas Nurtanio Bandung

Penulis sebelumnya telah memegang posisi penting di sejumlah bisnis yang beroperasi pada skala nasional dan dunia. penulis saat ini berprofesi sebagai praktisi bisnis (Pemilik Perusahaan di Bidang Fashion, Kuliner dan Bisnis Digital juga sebagai seorang akademisi. Penulis mengajar di Universitas Nurtanio Bandung, dan dosen tetap di perguruan tinggi indonesia mandiri (STMIK IM & STIE STAN IM). alumni dari Program Studi Informatika (S1) Sekolah Tinggi Manajemen Informatika dan Komputer Indonesia Mandiri (STMIK IM), Program Magister Manajemen (S2) di Pasca Sarjana Universitas Widyatama dan sedang melanjutkan studi Pendidikan (S3) Program Doktoral Pendidikan Teknologi dan Vokasional di Universitas Pendidikan Indonesia. Penulis aktif sebagai pembicara dan narasumber kewirausahaan digital juga berdedikasi sebagai praktisi bisnis / owner / pemilik beberapa perusahaan yang bergerak pada industri food, fashion and fun. serta aktif dalam organisasi nasional maupun internasional. Penulis juga seorang profesional dalam bidang public speaking & sebagai asesor kompetensi digital dan instruktur berpengalaman untuk sertifikasi kompetensi SKKNI BNSP Digital Marketing, Social Media Marketing dan Metodologi

Pelatihan Level III, Graphic Design serta Sertifikasi Kompetensi beberapa lainnya yang diselenggarakan dinas tenaga kerja kota bandung, PT Rice INTI dan Balai Besar Pengembangan Latihan Kerja (BBPLK) Bandung. Selain itu pula penulis produktif sebagai pendamping UMKM dan pengelola Inkubator Bisnis Perguruan Tinggi Indonesia Mandiri dengan membina dan melatih / coaching, mentoring serta melakukan pendampingan bisnis UMKM / Start UP untuk mahasiswa yang berminat menjadi seorang Entrepreneur / Technopreneurship tangguh yang mampu berdayasaing pada kancah internasional.