# BAB II

# LANDASAN TEORI



## Kursus Mengemudi

Lembaga Kursus adalah salah satu penyelenggara pendidikan diluar sekolah resmi (non-formal) untuk mengembangkan kemampuan dan keterampilan diri (Mahdy et al., 2021). Kursus Mengemudi secara spesifik dapat diartikan suatu pendidikan untuk mengembangkan kemampuan dan keterampilan diri dalam mengemudikan kendaraan khususnya mobil. KBBI sendiri mendeskripsikan kursus sebagai pelajaran tentang suatu pengetahuan atau keterampilan, yang diberikan dalam waktu singkat. Atau bisa juga diartikan sebagai lembaga di luar sekolah yang memberikan pelajaran serta pengetahuan atau keterampilan yang diberikan dalam waktu singkat.

## Rekayasa Perangkat Lunak

Secara bahasa, *Software Engineering* atau yang sering disebut dengan Rekayasa Perangkat Lunak dalam bahasa indonesia (RPL) tersusun dari 2 kata, *Software* dan *Engineering*. *Software* atau perangkat lunak bukan hanya sebuah program seperti yang diasumsikan banyak orang. Program adalah kode komputer yang dapat dieksekusi, yang tujuannya untuk melakukan satu atau lebih komputasi tertentu. Sedangkan *software* adalah kumpulan dari kode pemrograman komputer yang dapat dieksekusi, terorganisir dan terdokumentasi. Lalu, *Engineering* sendiri adalah semua hal yang meliputi pengembangan produk (baik fisik maupun digital) dengan menerapkan dan memanfaatkan prinsip-prinsip dan metodologi ilmiah yang tersusun dengan baik.

Sehingga, *Software Engineering* merupakan salah satu bagian dari *Engineering* yang berurusan dengan rekayasa perangkat lunak yang disusun secara baik dan menerapkan prinsip-prinsip, prosedur dan metodologi ilmiah. Hasil akhir dari RPL tentunya adalah perangkat lunak yang efisien, stabil dan bebas dari *bug*. Namun, apabila kita mengacu kepada pengertian menurut organisasi IEEE, RPL adalah (1) Penerapan dari sebuah pendekatan yang sistematik, disiplin, dan terukur untuk mengembangkan, mengoperasikan dan memelihara sebuah perangkat lunak; dalam hal ini penerapan rekayasa perangkat lunak. (2) Studi tentang pendekatan-pendekatan dari pernyataan diatas.

Selain IEEE, Fritz Bauer, ilmuwan komputer asal Jerman, mendefinisikan RPL sebagai “…pembentukan dan pemanfaatan prinsip teknik suara agar tercipta perangkat lunak yang ekonomis, yang stabil dan bekerja secara efisien di mesin sesungguhnya.”. Mengutip dari modul pembelajaran Rekayasa Perangkat Lunak – Pendekatan Terstruktur & Berorientasi Objek karya (Bahar et al., n.d.) Rekayasa perangkat lunak (*software engineering*) merupakan suatu disiplin ilmu yang membahas semua aspek produksi perangkat lunak (*software*), mulai dari tahap awal kajian spesifikasi / kebutuhan sistem sampai pemeliharaan sistem setelah digunakan (Sommerville, 2016). Pada definisi ini, ada dua istilah kunci: yang Pertama ‘Disiplin rekayasa’, yang berarti bahwa teknisi RPL membuat suatu alat bekerja. Mereka menerapkan teori, metode, dan alat bantu yang sesuai, selain itu mereka menggunakannya dengan selektif dan selalu mencoba mencari solusi terhadap permasalahan, walaupun tidak ada teori atau metode yang mendukung.

Selanjutnya teknisi RPL juga menyadari bahwa mereka harus bekerja dalam batasan organisasi dan keuangan, sehingga mereka berusaha mencari solusi dalam batasan-batasan ini. Kemudian yang kedua, ‘Semua aspek produksi perangkat lunak’, yang berarti Rekayasa Perangkat Lunak tidak hanya berhubungan dengan proses teknis dari pengembangan perangkat lunak tetapi juga dengan kegiatan seperti manajemen proyek perangkat lunak dan pengembangan alat bantu, metode, dan teori untuk mendukung produksi perangkat lunak. Bayangkan Rekayasa Perangkat Lunak seperti kita menuntun komputer untuk melakukan sebuah tugas sesuai dengan instruksi yang dituliskan sebelumnya. Teknisi RPL juga dapat diibaratkan sebagai arsitek di dunia digital, dimana mereka memanfaatkan wawasan ilmu komputer dan keahliannya untuk merancang, mengembangkan, menguji, dan meluncurkan aplikasi perangkat lunak.

## Sistem Informasi Berbasis Web

Sistem Informasi menurut Richard Vidgen adalah sebuah kumpulan komponen-komponen yang berinteraksi, komponen yang dimaksud adalah manusia, prosedur-prosedur, dan teknologi-teknologi yang ada, dimana komponen-komponen tersebut secara bersamaan mengumpulkan, memproses, menyimpan, dan mendistribusikan informasi untuk mendukung pengontrolan, pembuatan keputusan dan pengelolaan organisasi/perusahaan. Sistem informasi sendiri berisi informasi tentang organisasi/perusahaan terkait, contohnya, kondisi tentang operasional internal mereka, dan tentang lingkungan di dalam perusahaan tersebut, sebagai contoh informasi tentang para pelanggan, para supplier, dan kompetitor-kompetitor yang ada. Tanpa adanya sistem informasi, sebuah organisasi sulit untuk bertahan. Namun, bukan berarti bahwa sistem informasi harus menggunakan teknologi-teknologi informasi yang berbentuk komputer-komputer atau jaringan internet dan komunikasi, karena banyak sekali bentuk sistem informasi. Organisasi atau perusahaan sangat bergantung pada sistem informasi, meskipun aspek-aspek formal dari sistem informasi ini masih menggunakan sistem pengarsipan berupa kertas pada era sebelum adanya teknologi informasi itu sendiri.

Pengertian dari “teknologi” tidak bisa kita terjemahkan semudah yang dibayangkan. Teknologi bisa dan terkadang diaplikasikan ke perangkat-perangkat seperti komputer dan jaringan internet dan komunikasi, namun, juga bisa diaplikasikan ke praktik lapangan (contohnya : proses rekayasa perangkat lunak), dan teknik (contohnya : perancangan basis data). Sistem Informasi dapat diharapkan untuk memanfaatkan teknologi informasi, tetapi tidak berarti bahwa sistem informasi adalah teknologi. Sebuah sistem informasi pada intinya merupakan sebuah sistem yang melibatkan aktivitas manusia yang berada pada lingkup konteks organisasi/perusahaan. Tidak dapat dipungkiri, teknologi berperan sangat penting bagi sistem informasi, namun, tidak bisa kita mengabaikan peranan manusia dan dimensi-dimensi organisasi.

Fitzgerald (1998) menulis sebuah laporan yang membahas tentang krisis perangkat lunak, dimana rata-rata durasi pengerjaan untuk projek pemrograman sistem informasi (PSI) berkisar antara 18 bulan sampai dengan 5 tahun, kemudian, pada laporan tersebut dikemukakan juga bahwa, 68% dari keseluruhan projek melebihi jadwal yang ditetapkan, kemudian 75% dari keseluruhan projek harus dirancang ulang setelah proses implementasi dan 35% perusahaan setidaknya memiliki satu projek yang sudah ditinggalkan. Dapat disimpulkan bahwa projek pengembangan perangkat lunak merupakan sebuah projek yang sulit dan beberapa akan berpendapat bahwa solusi dari permasalahan ini adalah dengan melakukan pendekatan yang lebih baik dan lebih profesional pada saat proses pengembangan. Salah satu area yang coba mereka lakukan peningkatan adalah metodologi pengembangan perangkat lunak. Avison & Fitzgerald (2002) mendefinisikan metodologi sebagai berikut:

“Sebuah kumpulan prosedur, teknik, alat-alat dan bantuan dokumentasi yang dapat membantu pengembang sistem dalam rangkaian usaha mereka untuk mengimplementasi sebuah sistem informasi yang baru. Sebuah metodologi biasanya terdiri dari fase-fase, yang didalamnya akan terdapat sub-sub fase, yang akan mengarahkan pengembang sistem dalam memilih teknik yang paling sesuai pada setiap tahapan sebuah projek dan akan membantu mereka dalam merencanakan mengelola, mengontrol dan mengevaluasi proyek sistem informasi.”

Banyak sekali yang mendasari penggunaan metodologi untuk mengarahkan pemrograman sistem informasi. Menurut Fitzgerald (1996), alasan-alasan tersebut diantaranya adalah :

* Pembagian proses yang kompleks menjadi tugas-tugas yang dapat dikelola.
* Memfasilitasi kontrol dan pengelolaan projek. Salah satu peran pengelolaan adalah untuk mengelola risiko dan ketidakpastian.
* Kerangka kerja yang mengarahkan aplikasi teknik-teknik.
* Ekonomis, spesialisasi keahlian dan pembagian beban kerja.
* Secara hakikat, sebuah kerangka kerja untuk mengakuisisi dan melakukan sistematisasi pengetahuan. Sebuah metodologi harus mampu mendukung pembelajaran organisasi.
* Standarisasi, tim pengembang yang fleksibel, peningkatan produktivitas dan kualitas.

Namun, Fitzgerald memberikan daftar-daftar permasalahan yang dibawa oleh metodologi ini. Ada satu masalah mendasar tentang apa sebenarnya sebuah metodologi (anomali pengertian) dan sebuah tendensi bagi para pencipta metodologi untuk melakukan “perang metodologi”. Banyak metodologi yang minim dasar konseptual dan empiris, selain itu, ketika ada konsep yang mendasari metodologi tersebut cenderung berakar pada sebuah paradigma saintifik dan/atau teknik. Hal ini juga didukung oleh Wastell (1996) bahwa metodologi menjadi tujuan tersendiri :

“Metodologi menjadi sebuah *fetish,* sebuah prosedur yang digunakan dengan kekakuan patologis demi kepentingannya sendiri, bukan sebagai alat untuk mencapai sebuah tujuan. Dengan pemanfaatannya seperti saat ini, metodologi memberikan kesan meringankan terhadap keragu-raguan, metodologi ini mengisolasi pelaku pengembangan sistem informasi dari risiko-risiko dan ketidakpastian yang bisa diperoleh dengan melibatkan manusia dan masalah-masalah yang ada.”

Saat ini banyak bermunculan metode-metode pengembangan sistem informasi, metode-metode tersebut diantaranya yang paling populer adalah *Waterfall*, dan salah satu metode hasil pengembangannya *Rapid Application Development* (RAD), kemudian *Prototyping*, dan juga *Agile Software Development*, yang mencakup *Extreme Programming* (XP), SCRUM, dan *Lean Development*.

## Metode Rekayasa Perangkat Lunak *Prototyping*

Asosiasi antara *Prototyping* dan *Software Engineering* berawal saat *Software Engineering* sendiri baru saja dikemukakan antara tahun 1970-1980an. Metode ini menjadi salah satu yang terbaik diantara metode lain. Dibuktikan dengan banyaknya perangkat lunak yang dikembangkan dengan metode ini mendapatkan penerimaan yang baik di kalangan pengguna. Dan pengalaman pengguna (*User Experience*) terhadap penggunaan aplikasi atau sistem informasi yang dirancang dengan metode ini dianggap cukup baik. Berbeda dengan metode konvensional seperti *Waterfall*, dimana teknisi RPL harus menjelaskan terus menerus ke pemangku kepentingan dan calon pengguna tentang bagaimana bentuk aplikasi nantinya, metode *Prototyping* memvisualisasikan hasil perancangannya kepada dua belah pihak tersebut bahkan sampai melibatkan mereka pada tahap pengembangan untuk mendapatkan umpan balik agar produk yang dirancang dan dikembangkan oleh teknisi RPL sesuai dengan kemauan dan arahan mereka (pemangku kepentingan dan calon pengguna) (Bard, 2023).

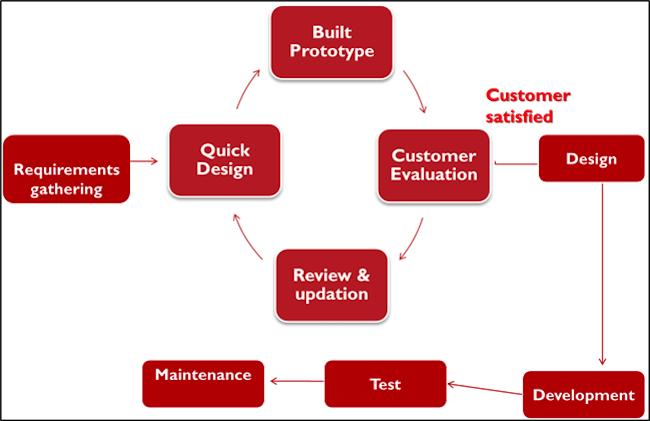
*Prototyping* menurut Kamus Webster 1913 adalah (produk) asli atau model (sederhana) yang kemudian disalin; pola apa pun yang akan diukir, atau disalin, dicetak atau sejenisnya (ke produk akhir). Kemudian sebagai pembanding, Dictionary.com mendefinisikan *prototype* sebagai (produk) asli atau model (sederhana) yang menjadi dasar atau bentuk sesuatu. Sedangkan Kamus Besar Bahasa Indonesia (KBBI) mendefinisikan prototipe sebagai, model yang mula-mula (model asli) yang menjadi contoh; contoh baku; contoh khas. Dari 3 sumber diatas, dapat kita cermati, ketiganya selalu menyebutkan ‘model’. Sehingga, dapat disimpulkan, *prototype* adalah sebuah model untuk memvisualisasikan akan seperti apa produk akhir nantinya.

Dalam *prototyping*, umumnya terdapat 3 tujuan yang ingin dicapai (Arnowitz et al., 2007), inovasi (Leonardo Da Vinci), penyempurnaan ide dan kebutuhan (Thomas Alfa Edison), serta komunikasi dengan pemangku kepentingan dan evaluasi (Henry Dreyfuss), tujuan-tujuan diatas dianggap masih relevan dengan kondisi dan keadaan saat ini, dapat kita artikan juga sebagai faktor yang mendorong penggunaan *prototyping* menjadi lebih canggih dan kompleks. Rekayasa perangkat lunak menggunakan metode *prototyping* seringkali dihadapkan dengan kebutuhan yang berbeda-beda yang belum tentu saling melengkapi satu sama lain bahkan hingga bertentangan. Berikut adalah 4 contoh kebutuhan yang bertolak belakang, yang pertama, perusahaan termotivasi untuk merilis produk yang terbaik dengan sesegera mungkin untuk mendapatkan keuntungan. Kedua, perusahaan dituntut untuk merilis sebuah produk dan fitur baru sebelum pesaing mereka melakukannya terlebih dahulu. Ketiga, terdapat risiko bahwa sebuah produk atau fitur baru ini bisa jadi tidak terlalu diinginkan oleh pengguna, walaupun dirilis tepat waktu dan sesuai dengan budget perusahaan. Keempat, terkadang pengguna tidak menginginkan produk atau fitur yang dianggap menguntungkan atau terbaik bagi perusahaan, sehingga terdapat konflik kepentingan antara kebutuhan pengguna dan keinginan perusahaan.

Garis besarnya disini adalah metode *prototyping* diharapkan mampu meminimalisir risiko-risiko terkait dengan rekayasa perangkat lunak dengan menciptakan sebuah model dari perangkat lunak yang dikembangkan sebelum sepenuhnya dirilis ke publik. Hal ini memungkinkan perusahaan untuk mendapatkan umpan balik dari pengguna dan pemangku kepentingan selama proses pengembangan dan menyumbangkan ide dan gagasannya kepada produk yang dikembangkan sesuai dengan keinginannya. Pada proses rekayasa perangkat lunak, *prototype* digunakan untuk berbagai tujuan, mulai dari membuat bayangan agar kemudian dikembangkan menjadi produk untuk digunakan oleh calon pengguna. Sebuah *prototype* dapat dimanfaatkan untuk menguji ide-ide kecil sampai besar.

Sebuah *prototype* yang dirancang sedini mungkin dapat dimanfaatkan untuk merencanakan inovasi, konsep fitur dan produk yang lebih mudah untuk ditangani. Dengan ini, *prototyping* memberikan kita penentuan lingkup yang lebih akurat, sehingga memungkinkan penghitungan waktu pengerjaan, penjadwalan pengerjaan, dan perencanaan biaya yang lebih baik. Pekerjaan diawal ini dapat membuat perencanaan proyek menjadi lebih baik. Sebuah *prototype* diawal yang memiliki tata letak tampilan, alur interaksi, status transisi, dan syarat kinerja dapat memberikan kita informasi yang dibutuhkan untuk mengukur penghitungan dan perencanaan kedepannya dengan tingkat kepercayaan diri yang tinggi.

Aktivitas *prototyping* memiliki ikatan historikal yang kuat dalam proses penemuan dan pembuktian dan perlindungan kekayaan intelektual. Meskipun pengertian *prototyping* tidak banyak berubah walaupun sudah lebih dari 90 tahun, *prototyping* berevolusi menjadi aktifitas canggih yang tujuannya tidak hanya untuk pengkonseptualan dan perancangan produk tetapi juga untuk strategi, definisi, spesifikasi, dan perencanaan sebuah produk. Lalu apa saja tahapan-tahapan *prototyping*?



Gambar 2. 1 Siklus Metode *Prototyping*

(Sumber: Hoang, 2022)

### Tahap 1 : *Requirements Gathering*

Tahapan awal metode *prototyping*, hampir serupa dengan metode-metode pengembangan perangkat lunak yang lain, yaitu pengumpulan kebutuhan. Kebutuhan sistem secara spesifik ditentukan pada tahapan ini. Pengguna sistem diberikan pertanyaan mengenai keseluruhan proses untuk selanjutnya digunakan oleh tim pengembang perangkat lunak sebagai acuan dan pengetahuan tentang apa yang diharapkan oleh pengguna terhadap sistem yang baru nantinya. Inti dari pelaksanaan tahap ini adalah untuk mengidentifikasi fungsionalitas dan kebutuhan sistem. Dapat diibaratkan juga seperti menggambar sketsa biru dari sebuah rumah–tanpa pemahaman yang mendasar tentang ruangan, corak, dan tata letak yang diinginkan pemilik rumah, proses konstruksi rumah tersebut menjadi sembarangan.

Jika kita melanjutkan perumpamaan membangun rumah diatas, tahap mengumpulkan kebutuhan sama dengan melakukan tanya jawab dengan pemilik rumah terkait rumah impian mereka. Analisa kebutuhan melibatkan interaksi bersama dengan calon pengguna dari perangkat lunak, dalam hal ini peserta dan penyedia jasa kursus mengemudi. Tim pengembang mengadakan sesi tanya jawab untuk mencatat ekspektasi dan masalah-masalah yang dihadapi pengguna. Proses pengumpulan data yang komprehensif ini tidak hanya mengungkap fitur-fitur yang diinginkan pengguna, namun juga mengidentifikasi tantangan-tantangan potensial yang mungkin akan dihadapi oleh pengguna. Dengan melakukan dokumentasi yang baik pada tahapan awal ini, tim pengembang dijamin akan membangun sistem sebuah sistem yang benar-benar menjembatani kebutuhan pengguna.

Pada tahap ini biasanya terdapat analisa proses bisnis yang saat ini berlangsung, kemudian kebutuhan pengguna, dan fitur-fitur dasar aplikasi.

### Tahap 2 : *Quick Design*

Tahap selanjutnya terdiri atas proses yang biasa dikenal dengan desain singkat. Selama tahapan ini, dasar rancangan sistem mulai terbentuk, rancangan ini tidak harus sebuah rancangan yang utuh dan lengkap, melainkan gambaran singkat tentang sistem nantinya. Secara singkat, tujuan dari pelaksanaan tahap ini adalah menerjemahkan kebutuhan pengguna yang teridentifikasi sebelumnya menjadi representasi sistem yang lebih jelas, meskipun sederhana. Ibarat sebuah sketsa awal dari seorang arsitek–bukan sebuah sketsa biru yang mendetail, tetapi sudah mewakili pokok dari struktur-struktur yang diinginkan dari bangunan nantinya.

Yang perlu ditekankan pada proses ini adalah kecepatan dan efisiensi. Tim pengembang diharap dapat memanfaatkan teknik-teknik desain singkat untuk membangun sebuah *prototype* dasar, sederhana, yang mewakili inti dari fungsionalitas aplikasi hasil terjemahan proses analisa kebutuhan sebelumnya. *Prototype* ini mungkin belum benar-benar berfungsi atau sudah berpenampilan indah, selama memenuhi tujuan pembuatan *prototype*–yaitu memberikan pengguna gambaran sepintas dari alur kerja dan rancangan sistem yang dibuat. Fokus dari tahap ini masih sama dengan tahap sebelumnya, yakni memperoleh informasi mengenai pengalaman pengguna selama ini, permasalahan-permasalahan potensial, dan menjamin *prototype* yang dihasilkan pada tahap ini sejajar dengan fitur yang dianggap pengguna penting.

Dengan memprioritaskan kecepatan dan umpan balik pengguna daripada detail-detail kecil selama tahap ini, tim pengembang bisa mengidentifikasi permasalahan potensial sedini mungkin. Hal ini memungkinkan koreksi arah pengembangan sebelum proses pengembangan sebenarnya dilakukan. Umpan balik yang didapat dari proses pengujian dengan pengguna atas *prototype* kasar yang dihasilkan menjadi bekal untuk memperbaiki rancangan dan memungkinkan perangkat lunak yang dihasilkan nantinya bersifat lebih baik dan berpusat pada pengguna.

Pada tahap ini, biasanya sketsa awal aplikasi beserta alur-alur kerja dari aplikasi mulai terbentuk.

### Tahap 3 : *Built* *Prototype*

Tahap ini adalah tahapan dimana *prototype* yang lebih lengkap dan detail mulai didesain menggunakan data yang didapat dari desain singkat sebelumnya. Alur aplikasi yang dibentuk pada tahap ini biasanya adalah sebagian kecil dari alur aplikasi pada produk akhir nantinya. Tim pengembang mulai memperluas desain *prototype* kasar dari tahap sebelumnya menjadi model yang lebih konkrit. *Prototype* ini ditujukan sebagai perwujudan fitur inti dari aplikasi, dimana pada akhirnya pengguna dapat berinteraksi dengan *prototype* ini dan memberikan umpan balik akan kemudahan dan efektifitas penggunaan fitur tersebut.

Kita bisa mengumpamakan tahapan ini seperti miniatur dari pembangunan sebuah gedung. Secara langsung bukan gedung yang asli, tapi para pihak terkait dapat mengetahui tata letak, ruangan-ruangan, dan alur masuk sampai keluar gedung. Tim pengembang memanfaatkan pengetahuan dari tahap sebelumnya untuk meningkatkan kualitas desain dan membuat model yang lebih baik lagi. Fokus dari tahap ini beralih dari *prototype* yang hanya merepresentasikan visual, tata letak, dan sebagainya, dikombinasikan dengan fitur-fitur dasar yang dapat berinteraksi oleh pengguna.

Model sederhana ini memungkinkan evaluasi lebih dalam terhadap kemudahan dan kegunaan aplikasi. Pengguna bisa menguji seberapa jelas atau mudah antarmuka yang ditampilkan, mengidentifikasi kemungkinan-kemungkinan navigasi aplikasi, dan memberikan umpan balik terhadap alur kerja secara keseluruhan. Masukan pengguna yang diterima ini berperan sebagai kritik untuk proses pengembangan nantinya, yang menjamin produk akhir nanti benar-benar memenuhi kebutuhan dan ekspektasi pengguna.

Pada tahap ini, iterasi *prototype* pertama dihasilkan, *prototype* yang murni berasal dari penerjemahan kebutuhan yang dilakukan tim pengembang.

### Tahap 4 : *Customer Evaluation*

Pada tahap ini, pengguna diberikan konsep sistem awal untuk dilakukan pengujian. Mengetahui model-model yang berfungsi dengan baik dan model-model yang tidak berfungsi dengan baik akan membantu tim pengembang nanti. Umpan balik pengguna pada tahap ini akan dikumpulkan, kemudian dikirimkan ke tim pengembang.

Tahap ini berfokus seputar validasi pengguna dan peningkatan iteratif. Dengan memiliki *prototype* yang dapat berfungsi pada tahap sebelumnya, tim pengembang mempresentasikannya ke klien, umumnya target pengguna atau pemangku kepentingan aplikasi. Hal ini berperan sebagai sebuah peluang kritis untuk memperoleh umpan balik berharga atas kelebihan dan kekurangan *prototype*.

Dapat kita misalkan seperti menampilkan draf kasar sebuah buku kepada calon pembaca. Penulis ingin mengukur tanggapan pembaca, memahami apa yang berhasil mempengaruhi emosi pembaca, dan mengidentifikasi area-area yang butuh ditingkatkan. Sama dengan apa yang terjadi pada tahap ini, pengujian dengan pengguna terhadap *prototype* yang berfungsi memungkinkan tim pengembang untuk menilai efektifitasnya secara langsung.

Melalui demonstrasi, panduan, dan interaksi pengguna dengan *prototype* tim pengembang secara aktif mengidentifikasi umpan balik klien. Umpan balik ini meliputi baik aspek-aspek positif–fitur-fitur yang disukai pengguna–dan area-area peningkatan–fitur-fitur yang kurang jelas, susah dinavigasi, atau yang tidak memenuhi ekspektasi pengguna. Dengan mengumpulkan dan menganalisa umpan balik ini, tim pengembang memperoleh pengetahuan yang dapat dijadikan panduan untuk memperbaiki *prototype* pada iterasi selanjutnya.

Pada tahap ini, biasanya berisi masukan-masukan dari klien, dalam hal ini, penyedia jasa kursus mengemudi, apakah *prototype* pertama aplikasi sudah memenuhi ekspektasi klien.

### Tahap 5 : *Review & Updation*

Jika pengguna merasa kurang senang dengan *prototype* saat ini, kita harus bisa meningkatkan *prototype* berdasarkan komen-komen dan rekomendasi yang diberikan oleh pengguna. Sampai semua kriteria pengguna terpenuhi, tidak dapat berlanjut ke tahapan selanjutnya. Sebuah sistem akhir diciptakan berdasarkan *prototype* akhir yang diterima sampai pengguna merasa puas dengan *prototype* yang dikembangkan.

Kita ibaratkan seorang pemahat patung menambahkan detail-detail yang memperindah model tanah liat nya berdasarkan umpan balik. Serupa dengan perumpamaan tersebut tim pengembang menganalisa umpan balik yang diterima selama pengujian dengan pengguna. Apabila aspek-aspek *prototype* gagal memenuhi ekspektasi pengguna, tim pengembang tidak akan melanjutkan fase pengembangan aplikasi dari sistem akhir. Mereka harus kembali lagi ke tahap desain singkat dan melakukan perbaikan *prototype* berdasarkan komen-komen dan rekomendasi pengguna. Tahap ini melibatkan proses penambahan detail untuk memenuhi ekspektasi pengguna dan menyederhanakan fitur-fitur yang membingungkan, meningkatkan antarmuka pengguna, atau mengkolaborasikan fitur-fitur baru yang dianggap berharga menurut pengguna.

Proses iteratif ini akan terus berlanjut sampai pengguna merasa puas dengan *prototype*. *Prototype* akhir ini, yang mewakili pokok-pokok umpan balik dan perbaikan pengguna berperan sebagai sketsa biru untuk proses pemrograman sistem akhir. Dengan memprioritaskan umpan balik pengguna selama siklus ini, model *prototype* menjamin bahwa produk akhir tidak hanya berfungsi, tapi benar-benar berfokus dengan pengguna dan menjawab kebutuhan yang diidentifikasikan pada tahap analisa kebutuhan sebelumnya.

Pada tahap ini, biasanya *prototype* hasil perbaikan dari tahap sebelumnya akan ditampilkan dan umpan balik dari pengguna terhadap *prototype*, dan yang terakhir adalah hasil keputusan terkait apakah pengguna sudah puas dengan hasil *prototype*.

### Tahap 6 : *Design*

Setelah menyelesaikan 4 tahap iteratif sebelumnya, tim pengembang selanjutnya akan menetapkan desain akhir. Tahap ini dilakukan untuk menerjemahkan *prototype* yang divalidasi oleh pengguna ke sketsa biru yang lebih mendetail dan komprehensif untuk proses pemrograman. Tidak sama dengan proses desain singkat sebelumnya, tujuan tahap ini beralih kepada perencanaan dan dokumentasi. Beberapa diagram-diagram UML akan dirancang untuk menampilkan fitur-fitur, struktur, dan perilaku aplikasi yang lebih jelas dan terstandarisasi.

Dengan merancang diagram-diagram UML, tim pengembang merumuskan *roadmap* komprehensif untuk pengembangan aplikasi nantinya. Diagram-diagram ini berperan sebagai alat untuk berkomunikasi, menjamin semua orang yang terlibat dalam proses pengembangan aplikasi memiliki pemahaman yang sama akan fitur-fitur, komponen, dan bagaimana aplikasi berinteraksi. Tahap desain akhir ini dapat menjembatani antara model *prototype* yang berfokus pada pengguna dengan proses pengembangan sesungguhnya. Diagram-diagram UML ini akan diterjemahkan menjadi fitur-fitur *prototype* yang tervalidasi menjadi bahasa teknis, membuka jalan bagi *programmer* untuk mulai melakukan pemrograman dan membangun sistem akhir.

Seperti yang disebutkan diatas, pada tahap ini akan menghasilkan *use case diagram, use case scenario, activity diagram,* dan *class diagram*.

### Tahap 7 : *Development*

Dengan tahap desain akhir sebelumnya sudah menghasilkan diagram-diagram UML, tahap pemrograman aplikasi dapat dimulai dengan lebih lancar. *Programmer* memanfaatkan sketsa biru yang komprehensif yang dihasilkan dari tahap sebelumnya yang harus diterjemahkan menjadi fitur-fitur yang tervalidasi pengguna menjadi aplikasi yang dapat bekerja. Tahap ini melibatkan proses pemrograman, mengintegrasikan macam-macam komponen, dan secara ketat melakukan pengujian terhadap sistem untuk menjamin aplikasi yang dihasilkan menaati spesifikasi-spesifikasi yang dirumuskan pada diagram-diagram UML sebelumnya. Proses pengembangan yang bersifat iteratif mendukung perbaikan-perbaikan *bug* berdasarkan hasil pengujian.

Setelah tahap pemrograman mencapai titik stabilitas dan fungsional, sistem selanjutnya diluncurkan ke lingkup produksi. Tahap ini melibatkan perencanaan dan eksekusi dengan cermat demi meminimalisir permasalahan yang mungkin dialami oleh pengguna nantinya. Dengan selesainya tahap pemrograman, tidak serta merta menandakan selesai nya siklus pengembangan perangkat lunak. Pengawasan yang terus berlanjut dan umpan balik pengguna terus dianggap krusial. *Programmer* secara aktif mendeteksi *bug* atau masalah-masalah yang muncul, dan selama sistem dapat ditingkatkan berdasarkan saran-saran pengguna dan kebutuhan yang berubah-ubah. Putaran umpan balik yang terus berlangsung ini menjamin bahwa aplikasi tetap relevan, ramah pengguna, dan secara efektif memenuhi kebutuhan pengguna dari waktu ke waktu.

Pada tahap ini, aplikasi sudah berbentuk dan dapat diakses pada web masing-masing pengguna, untuk selanjutnya dilakukan pengujian.

### Tahap 8 : *Test*

Tahap pengujian aplikasi pada metode pengembangan perangkat lunak *prototype* memiliki peran krusial untuk menjamin kualitas dan fungsionalitas aplikasi nantinya. Bukan hanya untuk mengidentifikasi *bug*, namun, juga untuk mengevaluasi kinerja sistem terhadap kriteria yang sudah dirumuskan. Pengujian mencakup berbagai macam aspek:

* **Pengujian Fitur**: Pengujian ini menjamin seluruh fitur yang dirancang pada diagram-diagram UML dan kebutuhan pengguna sudah diimplementasikan dengan benar.
* **Pengujian Kemudahan Penggunaan**: Pengujian ini menilai seberapa ramah sistem yang dihasilkan terhadap pengguna.

Dengan mengimplementasikan strategi pengujian yang komprehensif yang menjawab macam-macam aspek, tim pengembang dapat mengidentifikasi dan memperbaiki masalah sebelum sistem digunakan untuk masyarakat luas. Pengujian yang dilakukan ini menjaga kualitas produk akhir dan menjamin produk yang dihasilkan memenuhi standar tinggi yang dirumuskan selama proses *prototyping*.

### Tahap 9 : *Maintenance*

Setelah dilakukan pengujian secara komprehensif, demi mengurangi *lag* dan menghindari kerusakan yang parah, sistem harus dipelihara secara berkala. Proses ini bukan merupakan pendekatan reaktif untuk menunggu masalah-masalah muncul; melainkan strategi proaktif yang berfokus untuk mencegah masalah sebelum masalah tersebut mengganggu proses operasional pada sistem. Berikut adalah pemeliharaan sistem umumnya bekerja:

* **Mengidentifikasi dan Menjawab Masalah Potensial**: Memelihara aktivitas melibatkan pemeriksaan rutin dari kesehatan sistem, metrik kinerja sistem, dan kerentanan keamanan sistem. Pemeriksaan ini mampu mengungkap masalah-masalah potensial sedini mungkin, melakukan intervensi dengan tepat waktu sebelum masalah tersebut berkembang menjadi gangguan yang lebih besar.
* **Menerapkan Pembaruan**: Pembaruan perangkat lunak seringkali menjawab kerentanan dan *bug* yang diketahui. Penerapan secara berkala dari pembaruan ini menjamin sistem tetap aman dan terlindungi dari ancaman-ancaman yang muncul.
* **Mengoptimalisasi Kinerja**: Secara berkala, sistem mengakumulasikan penurunan kinerja data dan pengalaman. Pemeliharaan berkala meliputi tugas-tugas seperti optimalisasi data dan pembersihan sistem, menjamin sistem berjalan dengan lancar dan efisien.
* **Mencadangkan Pengujian dan Rencana Pemulihan Bencana**: Secara berkala mencadangkan prosedur pengujian dan rencana pemulihan bencana memverifikasi efektifitas rencana tersebut. Dengan melakukan hal ini, menjamin sistem mampu memulihkan fungsionalitasnya jika terjadi kejadian yang tidak terduga, meminimalisir hambatan dan kehilangan data.

## *Business Process Model and Notation* (BPMN)

*Business Process Model and Notation* adalah standar khusus untuk digunakan sebagai benchmark untuk pemodelan proses bisnis yang menghasilkan notasi grafis untuk memvisualisasikan proses bisnis (Firdaus et al., 2022). Notasi ini kemudian dapat kami konversikan menjadi *user flow* atau alur tugas yang dibutuhkan dalam melakukan proses *prototyping*. Notasi-notasi pada BPMN diantaranya adalah:

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Notasi** | **Nama** | **Deskripsi** |
| 1 |  | ***Start Event*** | Sesuai dengan nama notasinya, notasi ini mengindikasikan dimana sebuah rangkaian proses dimulai. |
| 2 |  | ***Intermediate Event*** | Notasi ini digunakan ketika terjadi sebuah proses diantara rangkaian proses. |
| 3 |  | ***End Event*** | Sebagaimana nama dari notasi disamping, notasi ini mengindikasikan akhir dari rangkaian proses. |
| 4 |  | ***Task*** | Sebuah tugas adalah aktivitas atomik yang dimasukkan di rangkaian proses bisnis. Sebuah tugas digunakan ketika sebuah pekerjaan dalam proses bisnis sudah tidak bisa lagi dipecah menjadi tugas yang lebih kecil |

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Notasi** | **Nama** | **Deskripsi** |
|  |  |  | lagi. |
| 5 |  | ***Choreograp-hy Task*** | Notasi ini merepresentasikan sebuah tugas yang melibatkan dua partisipan untuk menyelesaikannya. |
| 6 |  | ***Sub-process*** | Sebuah sub-proses adalah gabungan aktivitas yang dimasukkan pada sebuah proses. |
| 7 |  | ***Gateway*** | Sebuah gateway tidak sama dengan notasi kondisional pada flowchart, notasi ini digunakan untuk mengontrol pemecahan dan pertemuan pada sebuah rangkaian proses bisnis. Banyak macam dari gateway, lebih lanjutnya (OMG, 2011). |
| 8 |  | ***Sequence Flow*** | Notasi ini mengindikasikan sebuah alur yang tidak dimulai dari *Intermediate Event* yang terhubung ke sebuah tugas / sub-proses. |
| 9 |  | ***Message Flow*** | Notasi ini digunakan untuk menunjukkan alur informasi antara dua partisipan yang bersiap untuk mengirim dan |

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Notasi** | **Nama** | **Deskripsi** |
|  |  |  | menerima informasi tersebut. |
| 10 |  | ***Association*** | Notasi ini digunakan untuk menghubungkan informasi dan artifak terkait teks anotasi dan artifak lain yang terasosiasikan dengan elemen grafik lainnya. |

## *Unified Modeling Language* (UML)

Menurut buku dengan judul “*The Unified Modeling Language Reference Manual”* UML adalah tujuan umum dari bahasa pemodelan visual yang digunakan untuk menspesifikasi, memvisualisasikan, menyusun, dan mendokumentasi hal-hal yang terkait sistem perangkat lunak. UML mencatat semua keputusan dan pemahaman mengenai sistem-sistem yang wajib dibangun nantinya. Selain itu, UML digunakan untuk memahami, merancang, mengeksplorasi, mengkonfigurasi, memelihara, dan mengontrol informasi terkait sistem yang dikembangkan. Namun, UML bukan merupakan bahasa pemrograman, melainkan sebuah *tools* yang dapat menginspirasi pembuatan program yang selanjutnya bisa dikembangkan menggunakan bahasa pemrograman lainnya.

Tujuan dari UML sendiri diantaranya adalah UML dibuat sebagai tujuan utama untuk bahasa pemodelan dimana semua pelaku pengembangan perangkat lunak bisa menggunakannya. Bertujuan untuk mencakup konsep-konsep metode ternama yang nantinya mereka gunakan sebagai bahasa pemodelan. UML tidak dimaksudkan untuk dijadikan metode pengembangan yang lengkap, sebab UML tidak memiliki langkah-langkah mendetail tentang proses pengembangan perangkat lunak. Sekali lagi ditegaskan oleh (Jacobson et al., 2021) UML mencakup konsep-konsep yang dianggap penting untuk mendukung sebuah proses iteratif yang modern berdasarkan penerapan dengan arsitektur yang kuat untuk menyelesaikan kebutuhan berdasarkan masalah-masalah dan kasus-kasus yang dialami pengguna.

Yang terakhir, inti dari tujuan UML adalah pemodelan yang sesederhana mungkin selama masih mampu memenuhi syarat-syarat pemodelan sistem praktis secara penuh yang nantinya akan dibuat. Teknik-teknik pemodelan dari UML yang kita kenal diantaranya adalah *Use Case Diagram*, seperti yang kami jelaskan sebelumnya, kemudian terdapat *Activity Diagram*, *Class Diagram*, *Sequence Diagram*,kemudian banyak pemodelan-pemodelan lain yang kurang populer seperti *Statechart Diagram*, *Collaboration Diagram*, *Component Diagram*, *Deployment Diagram*, *Extensibility Construct*, dan pemodelan-pemodelan yang lain.

## *Use Case Diagram*

Menurut (Simanullang et al., 2021) pada jurnalnya dengan judul Sistem Informasi Pemesanan Menu Makanan Pada Rm Sedep Roso Rantauprapat Berbasis Web, peneliti sempat sedikit menjelaskan tentang istilah ini. *Use Case Diagram* adalah suatu pola atau gambaran yang menunjukkan kelakuan atau kebiasaan sistem. Sedangkan (Setiawansyah et al., 2022) menjelaskan bahwa *Use Case Diagram* adalah sebuah interaksi antara satu atau lebih aktor dengan sistem yang akan dibuat. *Use Case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem. Berikut adalah simbol-simbol yang digunakan pada Use Case Diagram:

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Notasi** | **Nama** | **Deskripsi** |
| 1 |  | ***Use Cases*** | Merepresentasikan fungsionalitas sebuah sistem, terkadang juga tujuan akhir dari aktor. Simbol ini selalu berada |
| **No.** | **Notasi** | **Nama** | **Deskripsi** |
|  |  |  | di dalam *Boundary Box*. |
| 2 |  | ***Actors*** | Aktor yang berinteraksi dengan sistem biasanya memicu *use case*. Simbol ini selalu berada di luar *Boundary Box.* |
| 3 |  | ***Association*** | Arah panah digunakan untuk menandakan sebuah hubungan antara aktor dan *use case* atau antara dua *use case*.  Panah dengan anotasi <<extend>> menandakan bahwa sebuah *use case* mungkin mengadopsi perilaku dari *use case* lain.  Panah dengan anotasi <<include>> menandakan bahwa sebuah *use case* menggunakan fungsionalitas *use case* lain. |
| 4 |  | ***Boundary Box*** | Simbol ini mengindikasikan batas lingkup sistem. |

## *Activity Diagram*

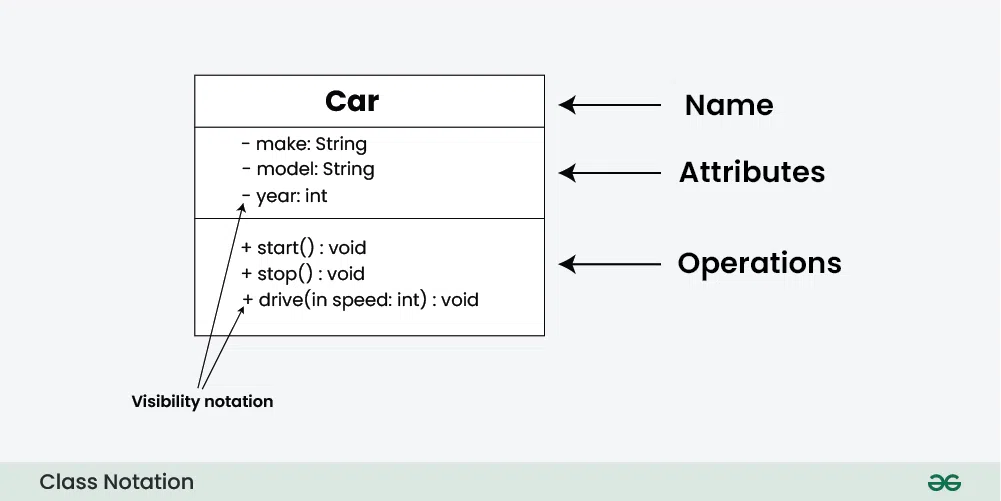
(Jacobson et al., 2021) menjelaskan *activity diagram* sebagai perwujudan khusus dari kondisi mesin yang ditujukan untuk memodel komputasi dan alur kerja sistem. Kondisi yang digambarkan pada *activity graph* mewakili kondisi ekseskusi komputasi yang dilakukan, bukan kondisi suatu objek secara spesifik. Umumnya, *activity graph* berasumsi bahwa komputasi yang terjadi tidak dipengaruhi oleh kejadian eksternal. *Activity diagram* biasanya berisi percabangan, lebih sering lagi percabangan kendali yang terbagi dua, dimana selanjutnya percabangan tersebut berjalan atau diproses bersamaan. Alur yang diproses secara bersamaan ini merepresentasikan aktivitas-aktivitas yang dapat dikerjakan oleh objek-objek atau orang-orang berbeda secara bersamaan pada sebuah organisasi. Seringkali, kejadian yang terjadi bersamaan ini muncul dari adanya agregasi, dimana objek memiliki proses yang harus dieksekusi secara bersamaan sendiri. Aktivitas yang terjadi secara bersamaan ini sebetulnya dapat dieksekusi secara bersamaan atau satu persatu. *Activity graph* sama halnya dengan *flowchart* yang sering kita kenal, perbedaan yang mendasar adalah *activity graph* memungkinkan terjadinya kendali proses secara bersamaan, selain kontrol sekuensial saja. Berikut adalah notasi-notasi yang ada pada *activity diagram*.

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Notasi** | **Nama** | **Deskripsi** |
| 1 |  | ***Initial state*** | Sebuah rangkaian proses hanya membutuhkan satu kondisi awal kecuali kita menggambarkan aktivitas yang terjadi didalam aktivitas. |
| 2 |  | ***Action* atau *Activity State*** | Sebuah aktivitas merepresentasikan eksekusi dari sebuah aksi atau objek oleh objek. |
| **No.** | **Notasi** | **Nama** | **Deskripsi** |
| 3 |  | ***Action Flow* atau *Control Flow*** | Alur aksi atau alur kontrol ini biasa digunakan untuk menunjukkan transisi dari satu kondisi aktivitas ke kondisi aktivitas lain |
| 4 |  | ***Decision Node* dan *Branching*** | Ketika kita akan membuat keputusan sebelum melanjutkan alur kontrol, kita bisa gunakan simpul keputusan. |
| 5 |  | ***Guard*** | Sebuah pelindung merujuk pada sebuah pernyataan yang ditulis disamping arah panah simpul keputusan. Pelindung membantu kita mengetahui batasan dan kondisi-kondisi yang menentukan alur sebuah proses |
| 6 |  | ***Fork*** | Simpul garpu digunakan untuk menggambarkan dua aktivitas yang terjadi secara bersamaan. Aktivitas yang berada dibawah simpul ini akan di eksekusi secara bersamaan. |

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Notasi** | **Nama** | **Deskripsi** |
| 7 |  | ***Join*** | Simpul gabung digunakan untuk melebur dua aktivitas yang sebelumnya diproses secara bersamaan menjadi satu proses utuh kembali. |
| 8 |  | ***Merge* atau *Merge Event*** | Penggabungan aktivitas ini dapat dilakukan apabila kita ingin menggabungkan dua atau lebih aktivitas yang sebelumnya tidak diproses secara bersamaan, namun harus diselesaikan sebelum aktivitas selanjutnya dapat dieksekusi. |
| 9 |  | ***Swimlanes*** | *Swimlane* digunakan untuk mengelompokkan aktivitas-aktivitas yang dieksekusi oleh objek atau orang yang sama. |
| 10 |  | ***Time Event*** | Notasi ini merujuk pada aktivitas yang membutuhkan waktu lebih lama untuk diselesaikan. |
| 11 |  | ***Final State* atau *End State*** | Kondisi akhir digunakan untuk menandakan akhir dari rangkaian proses. |

## *Class Diagram*

(Oktriwina, 2021) menjelaskan *class diagram* sebagai salah satu diagram struktur statis yang dimiliki UML dimana *class diagram* menggambarkan struktur sistem dengan menunjukkan sistem *class*, atribut-atribut yang dimiliki, metode-metode yang dapat dieksekusi, dan hubungan antar objek.(Jacobson et al., 2021) menjelaskan *class* itu sendiri adalah penjelasan dari konsep wewenang aplikasi atau solusi dari aplikasi. *Class* dapat dideskripsikan dari berbagai tingkatan presisi dan seberapa konkrit kita ingin mendeskripsikannya. Pada awal tahapan pengembangan perangkat lunak, *class diagram* seringkali berisi aspek-aspek logika dari masalah yang ingin diatasi. Sedangkan pada akhir tahapan pengembangan, *class diagram* dapat diisi keputusan-keputusan desain dan detail-detail implementasi di dalamnya. Berikut adalah contoh notasi sebuah *class*.



* **Nama *Class***: Biasanya ditulis di bagian atas kotak *class* dan ditulis tebal dengan perataan tengah.
* **Atribut** : Sering disebut dengan properti, merepresentasikan anggota-anggota data dari sebuah *class*. Biasanya berada di bawah nama *class* dan disertai oleh notasi visibilitas (contoh: [+] untuk *public*, [-] untuk *private*) dan tipe data.
* ***Methods*** : Dikenal juga sebagai fungsi atau operasi, merepresentasikan perilaku atau fungsionalitas sebuah *class*. Komponen ini berada di bawah atribut dan memiliki notasi visibilitas, jenis pengembalian, dan parameter-parameter setiap *methods*.
* **Notasi visibilitas** : Mengindikasikan tingkat aksesibilitas dari atribut atau *methods*. Notasi visibilitas umumnya ada 4 :
  + **[+]** untuk *public* (dapat diakses oleh semua *class*)
  + **[-]** untuk *private* (dapat diakses didalam *class* itu sendiri)
  + **[#]** untuk *protected* (dapat diakses oleh *subclass* dari *class* tersebut)
  + **[~]** untuk *package* atau *default visibility* (dapat diakses pada *class* dengan *package* yang sama)

Selain 4 komponen diatas, terdapat notasi tambahan yang mengindikasikan alur informasi antar *class* melalui *methods parameters* yang disebut dengan arah parameter. Notasi ini membantu menspesifikasi apakah parameter yang dimaksud merupakan masukan, keluaran, atau dua-duanya. Informasi ini dianggap krusial untuk memahami bagaimana data dikirimkan antar objek selama *methods* dieksekusi.

* ***In (input)*** : Parameter masukan adalah sebuah parameter yang dikirimkan dari objek yang memanggil (klien) ke objek yang dipanggil (server) selama pemanggilan *method*. Ditandai dengan arah panah yang mengarah ke *class* asal *method*.
* ***Out (output)*** : Parameter keluaran dikirimkan dari objek yang dipanggil (server) kembali ke objek yang memanggil (klien) setelah *method* dieksekusi. Ditandai dengan arah panah keluar dari *class* asal *method*.
* ***InOut (input* dan *output)*** : Parameter masukan/keluaran berperan sebagai masukan dan keluaran. Parameter ini membawa informasi dari objek yang memanggil ke objek yang dipanggil dan sebaliknya. Ditandai dengan arah panah yang mengarah dan keluar dari *class* asal *method.*

Pada *class diagram*, relasi antara *class* menggambarkan bagaimana setiap *class* terkoneksi atau berinteraksi dengan satu sama lain dalam sistem nantinya. Terdapat bermacam-macam jenis relasi pada pemodelan berorientasi objek, masing-masing memiliki tujuan spesifik. Berikut adalah jenis-jenis relasi pada *class diagram*:

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Notasi** | **Nama** | **Deskripsi** |
| 1 |  | ***Association*** | Sebuah relasi asosiasi dapat diartikan sebagai relasi dua arah antara dua *class*. |
| 2 |  | ***Directed Association*** | Relasi berarah dapat diartikan sebagai relasi dimana salah satu *class* tersebut memiliki ketergantungan oleh *class* yang lain |
| 3 |  | ***Aggregation*** | Relasi agregasi dapat diartikan sebagai relasi dimana satu *class* berisi atau tersusun dari *class* lain |
| 4 |  | ***Composition*** | Komposisi adalah bentuk relasi agregasi yang lebih kuat yang mengindikasikan bahwa satu *class* tidak bisa berdiri sendiri tanpa terkoneksi dengan *class* lain. |

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Notasi** | **Nama** | **Deskripsi** |
| 5 |  | ***Generalization (Inheritance)*** | Relasi ini merepresentasikan relasi dimana satu *class* (*subclass* atau *child*) mewarisi properti dan perilaku *class* (*superclass* atau *parent*) lain |
| 6 |  | ***Realization (Interface Implementation)*** | Relasi ini mengindikasikan bahwa sebuah *class* mengetahui operasi yang dilakukan oleh suatu antarmuka |
| 7 |  | ***Dependency Relationship*** | Relasi ketergantungan terwujud antar dua *class* ketika ada satu *class* bergantung pada *class* lain, tapi tingkat ketergantungan tidak sekuat relasi asosiasi atau relasi *inheritance* |
| 8 |  | ***Usage(Dependency) Relationship*** | Relasi menggunakan berarti satu *class* menggunakan atau bergantung pada *class* lain untuk mengeksekusi tugas-tugas tertentu atau mengakses fungsi-fungsi tertentu |

## *Hyper Text Markup Language* (HTML)

HTML adalah salah satu istilah pemrograman yang paling dikenal oleh masyarakat umum, namun, banyak perdebatan yang mengatakan bahwa sebenarnya HTML bukanlah sebuah bahasa pemrograman. Ada juga beberapa orang yang beranggapan bahwa karena dalam menulis HTML diperlukan setidaknya pemahaman dasar tentang pemrograman, maka HTML dianggap sebagai bahasa pemrograman. Mengacu dari jurnal yang ditulis oleh (Sari et al., 2022) HTML merupakan salah satu bahasa pemrograman yang digunakan untuk membuat website. HTML biasa ditulis untuk membantu perancangan struktur dasar halaman website atau bisa juga dianggap sebagai pondasi awal untuk menyusun kerangka halaman website secara lebih terstruktur sebelum masuk ke tahap desain dan fungsionalitas.

Terdapat pihak yang menyatakan bahwa sesungguhnya HTML merupakan sebuah bahasa *markup*, disebut demikian karena penulisan HTML didominasi oleh penulisan *tag* atau tulisan di dalam tanda kurung lancip (<html>) dimana masing-masing *tag* yang kita tulis nantinya akan dibaca kemudian dikonversi menjadi tampilan website yang kita baca setiap harinya. Singkatnya, HTML hanyalah kumpulan dari berbagai macam *tag* (seperti: <h1>, <p>, <div>, dll.) yang bisa kita gunakan untuk membangun sebuah halaman website dan masing-masing tag tersebut hanyalah sebuah cara untuk kita menjelaskan kepada mesin konten apa yang ingin kita tampilkan.

## *Cascading Style Sheets* (CSS)

Pada era modern saat ini, hampir tidak bisa kita temui rangkaian kode HTML tanpa dilengkapi CSS. Menurut (Sari et al., 2022) CSS adalah bahasa pemrograman yang ditujukan untuk memberikan modifikasi tampilan elemen-elemen web seperti *font, outline, background,* menyesuaikan tampilan website dengan ukuran layar, dan sebagainya. Jika HTML digunakan untuk menempatkan konten-konten apa saja yang ingin ditampilkan pada sebuah halaman web, CSS digunakan untuk memberikan pemahaman kepada mesin untuk melakukan modifikasi terhadap tampilan elemen dan penataan tata letak lebih lanjut.

Karena sejatinya, HTML tidak dirancang untuk menentukan aspek visual pada sebuah desain website. Sebab, fokus utama dari HTML adalah membagi struktur sebuah halaman website. Oleh karena itu, dikenalkan skrip “pendamping” untuk memperindah tag-tag HTML yaitu CSS. Selain itu, tujuan penggunaan CSS adalah untuk memberikan kesan konsisten di seluruh website.

## *Tailwind* CSS *Framework*

(Somi, M., 2023) dalam jurnalnya menjelaskan bahwa *Framework* *Tailwind* CSS adalah *framework* CSS yang mengutamakan penggunaan kelas utilitas yang paling populer dan bertujuan untuk membangun tampilan antarmuka khusus dengan cepat dan mudah. Maksudnya, berbeda dengan *Bootstrap* yang tergolong *framework* UI *kits*, *tailwind* tidak menyediakan komponen-komponen siap pakai. *Tailwind* tidak mempunyai tema bawaan. Dengan *tailwind*, kita memberikan *style* dengan mengetikkan kelas-kelas yang sudah ditentukan sebelumnya ke kodingan HTML yang kita kerjakan.

Lebih lanjut lagi, (Somi, M., 2023) menjelaskan bahwa dengan menggunakan *tailwind*, memberikan kita kemampuan untuk mempercepat proses pemrograman tanpa kita harus menulis kode CSS di *file* lain melainkan, menulisnya secara bersamaan di kode markup HTML. Selain itu, *tailwind* memberikan kita kemampuan untuk melakukan kustomisasi secara penuh sesuai keinginan kita. Efek samping dari menggunakan *tailwind* adalah kode HTML yang dihasilkan akan jauh lebih panjang

## *Javascript*

Halaman website yang dihasilkan dari hanya menggunakan bahasa HTML & CSS cenderung statis dan kurang menarik. Untuk membuat tampilan yang lebih dinamis diciptakan sebuah bahasa pemrograman baru demi mengatasi kekurangan ini, yakni *Javascript*. Sebagai referensi, (Noviantoro et al., 2022) menjelaskan bahwa *Javascript* adalah salah satu bahasa pemrograman yang digunakan untuk dapat berjalan di web browser. Pada awal pengembangan bahasa pemrograman ini sempat disebut dengan nama *Mocha*, kemudian berubah penamaannya menjadi *Live-Script*, dimana ketika masa rilis, diubah lagi menjadi *Javascript*. Lebih jauh lagi dijelaskan bahwa *Javascript* adalah *script* program berbasis *client* yang dieksekusi oleh browser sehingga membuat halaman web melakukan tugas-tugas tambahan yang tidak bisa dilakukan hanya dengan memanfaatkan HTML biasa. Selain alasan yang kami sebut diatas, beberapa interaksi yang ingin kami munculkan pada aplikasi ini tidak dapat diselesaikan hanya dengan menggunakan HTML dan CSS.

## *Framework* PHP Laravel

Sebelum kita lebih jauh membahas tentang *framework Laravel*, akan kami jelaskan secara singkat apa yang dimaksud dengan PHP, karena istilah ini juga bukan bagian yang terpisahkan dari perancangan dan pengembangan sistem informasi atau perangkat lunak berbasis web. PHP atau *Hypertext Preprocessor* sebagaimana yang dijelaskan (Adrianto S., 2021) adalah sebuah bahasa pemrograman untuk membuat web yang bersifat *server-side* *scripting*, PHP memungkinkan untuk membuat halaman web bersifat dinamis. Selain itu, PHP membutuhkan *Database Management System* (DBMS) untuk dijalankan secara bersamaan. DBMS yang paling populer di kalangan pelajar pemrograman salah satunya adalah MySQL, namun, PHP juga mendukung DBMS lain seperti Oracle, Microsoft Access, Interbase, D-Base, PostgreSQL, dan DBMS yang lainnya.

Satu hal yang mungkin menjadi pertanyaan adalah apabila PHP dikenal dengan *Hypertext Preprocessor*, lalu apa kepanjangan PHP? Menurut tulisan yang diterbitkan melalui web sekawanstudio.com yang ditulis oleh (Miranda R.A., 2023) Pada tahun 1994, ketika Rasmus Lerdorf pertama kali menemukan *hypertext preprocessor*, beliau menggunakannya untuk memantau jumlah pengunjung atau yang sering kita sebut dengan *traffic website* dari halaman web pribadi nya atau dalam bahasa inggris dikenal dengan *Personal Home Page*. Alasan tersebutlah yang menyebabkan bahasa pemrograman ini dijuluki sebagai PHP.

Menurut penjelasan dari (Subecs, 2021) Laravel merupakan *framework* PHP yang paling sering digunakan untuk *programmer* pemula dan berpengalaman. Laravel dianggap mampu mengurangi durasi pengembangan sistem perangkat lunak serta mempersiapkan pasar dengan metode PHP berorientasi objek yang lebih modern. *Syntax-syntax* ekspresif dan *function-function* modern yang dimiliki Laravel disukai oleh para *programmer* yang ingin mengembangkan web atau aplikasi yang lebih kompleks. Dengan menggunakan *framework* ini diyakini dapat mempermudah proses pengembangan karena Laravel menggunakan sistem paket modular dimana modul-modul yang disediakan saling terkait satu sama lain, dimana kita bisa mengembangkan sistem perangkat lunak yang lebih luas lagi. *Framework* ini memberikan kita jalan pintas yang memungkinkan *programmer* berkonsentrasi terhadap masalah-masalah yang lebih penting.

Lebih lanjut (Subecs, 2021) menyebutkan fitur-fitur atau kelebihan-kelebihan yang dimiliki Laravel yang menjadikan sebab mengapa banyak *programmer* menyukai *framework* ini. Kelebihan-kelebihan tersebut diantaranya adalah:

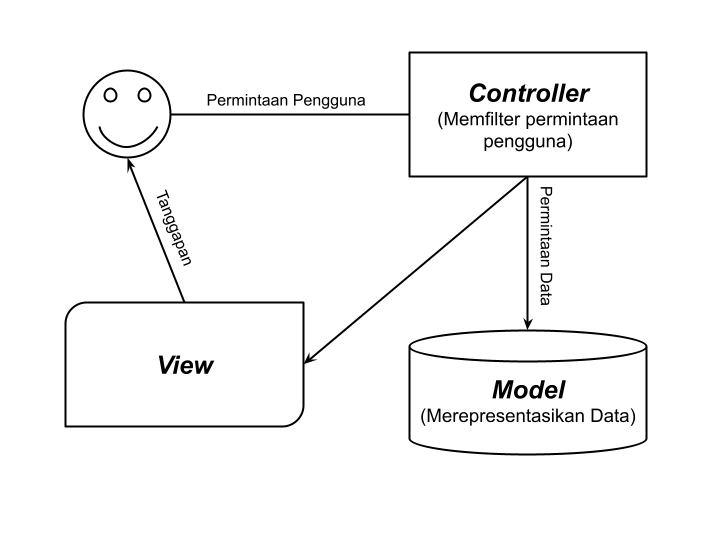


### Routing

Fungsi utama dari sebuah *framework* sistem perangkat lunak adalah untuk menerima permintaan dari pengguna dan mengirim tanggapan mereka biasanya melalui HTTP(S). Artinya bahwa tugas pertama selama proses penerapan ini ialah menentukan rute-rute yang dibutuhkan. Tanpa rute, sebuah sistem tidak akan mampu mencapai pengguna. Rute itu sendiri pada dasarnya merupakan sebuah URI (*Uniform Resource Identifier*) yang memungkinkan komunikasi dengan dunia luar dengan alamat URL yang diketahui. Pada dasarnya *client* mengirimkan sebuah permintaan untuk mendapatkan rute yang ditentukan, sesuai dengan kebutuhan, meneruskannya ke sebuah *controller* yang memproses permintaan tersebut. Folder */routes* yang berada di dalam folder *root* projek yang berisi *file-file* rute tersebut.

### Arsitektur MVC

Laravel dibangun diatas pondasi arsitektur MVC. *Model-View-Controller (MVC)* merupakan sebuah pola perancangan yang mencakup tiga bagian utama: *model, view,* dan *controller.* Komponen-komponen ini sudah dikonfigurasi untuk menangani aspek-aspek pengembangan khusus dari aplikasi. Komponen *View* ditujukan untuk menangani logika tampilan pengguna. Komponen *Controller* menerima input data dan memproses input tersebut. Komponen *Controller* ini bekerja sebagai antarmuka antara komponen *model* dan *view.* Sedangkan komponen *Model* adalah logika yang memiliki koneksi dengan data-data yang terkait user. Komponen ini merupakan komponen utama dari pola arsitektur MVC dan merepresentasikan data-data yang ditransaksikan antara *view* dan *controller*.



### *Views* dan *Templates*

*Views* bertanggung jawab untuk menampilkan tanggapan dari *controller* sesuai dengan format yang tepat, biasanya sebagai halaman website. *Views* dan relasinya dengan *Controller* ini bisa dikembangkan dengan mudah menggunakan bahasa *Blade template* atau lebih sederhana lagi dengan skrip-skrip PHP biasa. *View* adalah objek yang paling umum kembali ke rute-rute. *View* mendapatkan data dari rute-rute atau *controller* dan menyisipkannya ke sebuah template, sehingga mereka membantu satu sama lain untuk memisahkan logika bisnis dari logika presentasi.

### *Controller*

Pada sebuah *framework MVC*, *Controller* sebagaimana namanya, mengontrol alur data antara dataset-dataset dan *views*. Hal ini merupakan mekanisme yang membawa pengguna-pengguna ke lapisan presentasi. *Controller* menerima permintaan, memproses mereka dan mengirimkan tanggapan yang sesuai. *Controller* menangani tugas-tugas seperti menarik data dari *database*, menangani penerimaan data dari form-form dan menyimpan data ke *database*. *Controller* biasanya berada di folder *app/http/controllers*.

### Berinteraksi dengan *database*

Ada 3 proses interaksi dengan *database* yang terbaik dari *framework* Laravel menurut (Subecs, 2021):

* *Migration*

Migrasi adalah skrip-skrip PHP yang digunakan untuk memanipulasi struktur dan konten *database*. Seorang *programmer* yang bekerja dalam sebuah tim diharapkan untuk mampu dengan baik mensinkronisasi tugas-tugas *database* dengan rekan-rekan *programmer* lain. Dalam hal ini, migrasi beroperasi sebagai sebuah *version control*. Migrasi-migrasi ini dikirimkan bersama dengan detail waktu migrasi, sehingga migrasi ini dieksekusi berdasarkan urutan yang benar. Dengan memanfaatkan migrasi ini memungkinkan *database* memiliki struktur keadaan yang konsisten.

* *Database Seeding*

Dalam *framework* Laravel, kita bisa mengisi *database* dengan pembantu pengisian *database*, daripada melakukannya dengan manual. Dengan melakukan pengisian ini, kita bisa meng-*upload* data pengujian ke *database* menggunakan sebuah cara sederhana. Kita mampu mencari *file-file* pengisian ini di folder *database/seeders*.

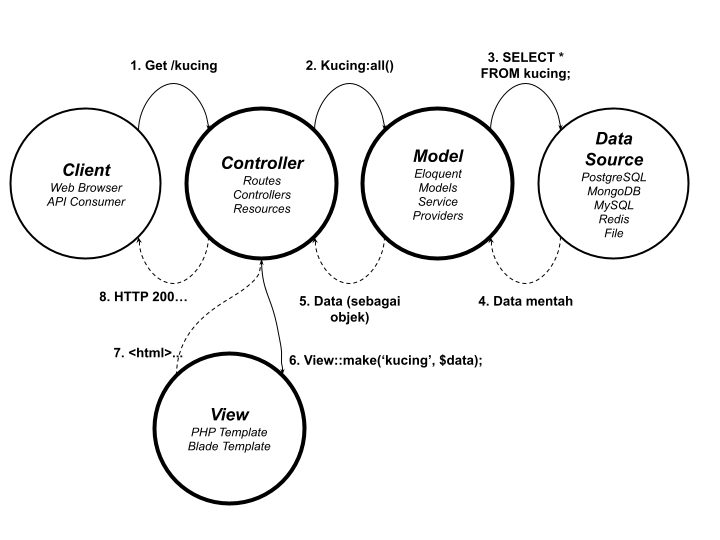
* *Query Builder*

Laravel memberikan kita *query builder* yang mudah digunakan yang ditujukan untuk membuat dan mengeksekusi *query-query database*. *Programmer* dapat menggunakannya untuk bermacam-macam operasi *database* dan bekerja dengan sempurna dengan sistem-sistem *database* yang didukung Laravel.

### *Model* dan *Eloquent*

*Model* seringkali dihubungkan dengan sumber-sumber informasi aplikasi dan seringkali digunakan bersama dengan rekaman-rekaman *database*. *Model* ini merupakan kelas-kelas yang merepresentasikan entitas-entitas di dalam aplikasi, seperti pengguna, artikel berita, atau *event-event*. Laravel meluncurkan sebuah fitur yang disebut dengan *Eloquent*, yang merupakan pemetaan hubungan antar objek yang efisien dimana para *programmer* dapat menentukan entitas, kemudian memasukkannya ke tabel-tabel *database* terkait dan memanfaatkan metode-metode PHP untuk mengeksekusi *entry* entitas-entitas tersebut, daripada melakukan *statement* SQL secara mentah. Dengan fitur ini, kita bisa menjalankan *query* database ORM dengan efisien, tanpa menuliskan instruksi-instruksi SQL apapun. *Eloquent* berjalan sebagai sebuah lapisan-model pada aplikasi yang dikembangkan.

Berikut adalah ilustrasi bagaimana interaksi yang terjadi dari masing-masing komponen ketika diberikan perintah untuk menampilkan semua data yang ada pada tabel kucing.



Kita ketahui *client* (melalui *web browser* atau *API Consumer*) ingin menampilkan semua data yang tersimpan pada tabel kucing. Perintah ini diteruskan menuju *Controller* terlebih dahulu untuk diproses permintaan data *client* kemudian menentukan *routes* sumber pengambilan data. Selanjutnya *Model* menangkap perintah pengambilan data dari tabel kucing dan mengirimkannya ke sumber data. Selanjutnya, data yang diambil dari sumber data ini dikembalikan lagi ke *Model* untuk diperiksa apakah data yang diambil hasilnya sesuai dengan perintah yang diberikan, selanjutnya karena Laravel mengusung konsep OOP, data ini ditangkap sebagai objek oleh *Controller*, yang kemudian dibawa ke *View* untuk diberikan skrip PHP yang selanjutnya disisipkan kedalam kode HTML untuk dikembalikan lagi ke *Controller* agar ditampilkan ke *Client*.

### *Authentication* dan *Authorization*

Proses pendaftaran dan *login* pengguna pada sistem perangkat lunak adalah suatu fitur yang umum dan penting. Laravel memberikan *tools-tools* berbeda untuk memanfaatkan fungsi-fungsi yang dibutuhkan pada proses tersebut menjadi lebih aman dan mudah. Disamping sistem autentikasi dari Laravel, kita bisa menggunakan metode sederhana untuk proses otorisasi operasi pengguna ketika mereka akan mengakses sebuah sumber data. Meskipun pengguna sudah sah secara autentikasi, tetapi belum tentu mereka berhak membaca atau mengubah isi beberapa model-model *Eloquent* yang ditangani aplikasi atau rekaman rekaman *database*.

### *Middleware*

*Middleware* dimanfaatkan untuk mem-*filter* permintaan HTTP pada saat pengguna / *client* memasuki aplikasi. *Programmer* dapat menggunakan *middleware* untuk lakukan tugas-tugas berikut ini: pemeriksaan autentikasi dan otorisasi, validasi sesi dan modifikasi variabel sesi, membaca, mengatur atau memodifikasi *header* permintaan dan tanggapan; merekam transaksi data; pemanggilan API, dan lain-lain.

### *Validation*

Kelas validatorLaravel membantu memvalidasi fungsionalitas elemen-elemen seperti form, model *database*. Validator ini mendukung penerimaan input data; mendeklarasikan peran-peran validasi khusus dan mendeklarasikan pesan-pesan validasi khusus.

### Pengamanan aplikasi

Sebelum meluncurkan sistem perangkat lunak kita ke lingkungan publik, *programmer* harus mempertimbangkan beberapa pertanyaan-pertanyaan tentang keamanan. Laravel melindungi sistem perangkat lunak terhadap serangan-serangan yang sering terjadi. Beberapa diantaranya adalah: *Cross-site request forgery* (CSRF), *Cross-site scripting* (XSS), SQL *injection*, *mass assignment vulnerability.*