BAB III

METODOLOGI PENELITIAN

3.1 Metode Penelitian

Metode penelitian yang digunakan adalah metode penelitian deskriptif dan komparatif. Menurut Erliana (2008:20), penelitian deskriptif adalah penelitian terhadap fenomena atau populasi tertentu yang diperoleh oleh peneliti dari subjek berupa individu, organisasional, industri atau perspektif yang lain. Penelitian deskriptif dilakukan untuk menjawab pertanyaan tentang: apa, siapa, kapan, di mana, dan bagaimana yang berkaitan dengan karakteristik populasi atau fenomena tersebut. Penelitian juga dilakukan dengan menggunakan metode komparatif yang dilakukan dengan membandingkan teori yang ada dengan praktik yang ditemui di dalam perusahaan dan menarik kesimpulan.

3.2.1 Desain Penelitian

Desain penelitian adalah bagian dari perencanaan penelitian yang menunjukkan usaha peneliti dalam melihat apakah penelitian yang direncanakan telah memiliki validitas internal dan eksternal yang komprehensif. Desain atau perancangan dalam pembangunan perangkat lunak merupakan upaya untuk mengonstruksi sebuah system yang memberikan kepuasan (mungkin informal) akan spesifikasi kebutuhan fungsional, memenuhi target, memenuhi kebutuhan secara implisit atau eksplisit dari segi performansi maupun penggunaan sumber daya, kepuasan batasan pada proses desain dari segi biaya, waktu, dan

perangkat. Kualitas perangkat lunak biasanya dinilai dari segi kepuasan penggunaan perangkat lunak terhadap perangkat lunak yang digunakan.

3.2.2 Jenis dan Metode Pengumpulan Data

Jenis dan metode pengumpulan data yang digunakan dalam perancangan Aplikasi Panduan Shalat Sunnah dan Kumpulan Do'a Berbasis Android yaitu dari sumber data primer yang terdiri dari wawancara dan observasi, serta sumber data sekunder yaitu dari dokumentasi.

3.2.2.1 Sumber Data Primer

1) Wawancara : proses mengumpulkan data dengan melakukan Tanya jawab langsung mengenai apa yang dibutuhkan dengan narasumber yang memahami objek yang diteliti.
2) Quesioner : proses pengumpulan data yang dilakukan cara melakukan penyebaran questioner kepada sample responden.

3.2.2.2 Sumber Data Sekunder

Dokumentasi : proses pengumpulan data dengan mencari data-data yang berhubungan dengan objek yang diteliti.

3.2.3 Metode Pendekatan dan Pengembangan Sistem

Dalam membangun suatu aplikasi diperlukan suatu pendekatan dan pengembangan sistem yang akan menentukan proses penyelesaian rekayasa perangkat lunak dan pendekatan sistem yang digunakan pada Aplikasi Panduan Shalat Sunnah dan Kumpulan Do'a Berbasis Android

yaitu pendekatan berorientasi objek dan pengembangan sistem dengan menggunakan model prototype.

3.2.3.1 Metode Pendekatan Sistem

Sistem adalah suatu jaringan kerja dari prosedur-prosedur yang saling berhubungan untuk melakukan suatu kegiatan yang merupakan rangkaian dari kerja komponen elemen dimana menghasilkan suatu tujuan yang hendak dicapai. Sistem yang digunakan harus dapat mencapai sasaran yang sesuai. Pendekatan sistem merupakan jaringan kerja dari prosedur yang lebih menekankan urutan-urutan operasi dalam sistem.

Metode pendekatan sistem yang digunakan pada Aplikasi Panduan Shalat Sunnah dan Kumpulan Do'a Berbasis Android yaitu Pendekatan berorientasi objek merupakan suatu teknik atau cara pendekatan dalam melihat permasalahan dan system (system perangkat lunak, system informasi, atau system lainnya). Pendekatan berorientasi objek akan memandang system yang akan dikembangkan sebagai suatu kumpulan objek yang berkorespondensi dengan objek-objek dunia nyata. Ada banyak cara untuk mengabstraksikan dan memodelkan objek-objek tersebut,mulai dari abstraksi objek, kelas, hubungan antar kelas sampai abstraksi system. Saat mengabstraksikan dan memodelkan objek ini, data dan proses-proses yang dipunyai oleh objek akan dienkapsulasi (dibungkus) menjadi satu kesatuan.

Dalam rekayasa perangkat lunak, konsep pendekatan berorientasi objek dapat diterapkan pada tahap analisis, perancangan, pemrograman dan pengujian perangkat lunak. Ada berbagai teknik yang dapat digunakan pada masing-masing tahap tersebut dengan aturan dan alat bantu pemodelan tertentu. Sistem berorientasi objek merupakan sebuah system yang dibangun dengan berdasarkan metode berorientasi objek adalah sebuah system yang komponennya dibungkus (dienkapsulasi) menjadi kelompok data dan fungsi. Setiap komponen dalam system tersebut dapat mewarisi atribut dan sifat dan komponen lainnya, dan dapat berinteraksi satu sama lain. Adapun karakteristik atau sifat-sifat yang dipunya sebuaah system berorientasi objek adalah sebagai berikut :

1. Abstraksi : prinsip untuk merepresentasikan dunia nyata yang kompleks menjadi satu bentuk model yang sederhana dengan mengabaikan aspek-aspek lain yang tidak sesuai dengan permasalahan.
2. Enkapsulasi : pembungkusan atribut data dan layanan (operasi-operasi) yang dipunyai objek untuk menyembunyikan implementasi dan objek sehingga objek lain tidak mengetahui cara kerjanya.
3. Pewarisan (*inheritance)* : mekanisme yang memungkinkan satu objek mewarisi sebagian atau seluruh definisi dan objek lain sebagai bagian dari dirinya.

4. *Reusability* : pemanfaatan kembali objek yang sudah didefinisikan untuk suatu permasalahan padan permasalahan lainnya yang melibatkan objek tersebut.
5. *Generalisasi dan Spesialisasi* : menunjukan hubungan antara kelas dan objek yang umum dengan kelas dan objek yang khusus.
6. Komunikasi Antar Objek : komunikasi antar objek dilakukan lewat pesan (*class*) yang dikirim dan satu objek ke objek yang lain.
7. *Polymorphism* : kemampuan suatu objek untuk digunakan dibanyak tujuan yang berbeda dengan nama yang sama sehingga menghemat baris program.

3.2.3.2 Metode Pengembangan Sistem

Metode yang digunakan untuk pengembangan sistem yaitu metode *prototyping*. *Prototype* merupakan salah satu metode pengembangan perangat lunak yang banyak digunakan. Dengan metode prototyping ini pengembang dan pelanggan dapat saling berinteraksi selama proses pembuatan sistem. Sering terjadi seorang pelanggan hanya mendefinisikan secara umum apa yang dikehendakinya tanpa menyebutkan secara detal output apa saja yang dibutuhkan, pemrosesan dan data-data apa saja yang dibutuhkan. Sebaliknya disisi pengembang kurang memperhatikan *efesiensi algoritma*, kemampuan sistem operasi dan *interface* yang menghubungkan manusia dan komputer. Untuk mengatasi

ketidakserasian antara pelanggan dan pengembang , maka harus dibutuhakan kerjasama yang baik diantara keduanya sehingga pengembang akan mengetahui dengan benar apa yang diinginkan pelanggan dengan tidak mengesampingkan segi-segi teknis dan pelanggan akan mengetahui proses-proses dalm menyelasaikan system yang diinginkan. Dengan demikian akan menghasilkan sistem sesuai dengan jadwal waktu penyelesaian yang telah ditentukan. Paradigma dari metode *prototyping* adalah sistem informasi yang menggambarkan hal-hal penting dari sistem informasi yang akan datang. *Prototipe* sistem informasi bukanlah merupakan sesuatu yang lengkap, tetapi sesuatu yang harus dimodifikasi kembali, dikembangkan, ditambahkan atau digabungkan dengan sistem informasi yang lain bila perlu. Ada empat langkah karakteristik metode *prototyping* yaitu:

1. Pemilahan fungsi . Mengacu pada pemilahan fungsi yang harus ditampilkan oleh *prototyping*. Pemilahan harus selalu dilakukan berdasarkan pada tugas-tugas yang relevan yang sesuai dengan contoh kasus yang akan diperagakan
2. Penyusunan Sistem Informasi. Bertujuan untuk memenuhi permintaan akan tersedianya *prototype*
3. Evaluasi
4. Penggunaan Selanjutnya

Sedangkan jenis-jenis *prototyping* yaitu :

1) *Feasibility prototyping* – digunakan untuk menguji kelayakan dari teknologi yang akan digunakan untuk system informasi yang akan disusun.
2) *Requirement prototyping* – digunakan untuk mengetahui kebutuhan aktivitas bisnis user.
3) *Desain Prototyping* - digunakan untuk mendorong perancangan system informasi yang akan digunakan.
4) *Implementation prototyping* – merupakan lanjytan dari rancangan *protipe, prototype* ini langsung disusun sebagai suatu system informasi yang akan digunakan.

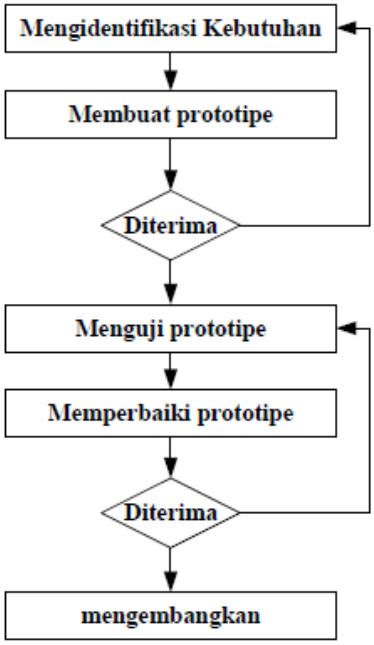
Keunggulan dari metode *prototyping* yaitu :

1. *End user* dapat berpartisipasi aktif
2. Penentuan kebutuhan lebih mudah diwujudkan karena pelanggan berperan aktif dalam pengembangan sistem
3. Lebih menghemat waktu dalam pengembangan system

Sedangkan kelemahannya yaitu :

1. Proses analisis dan perancangan terlalu singkat
2. Mengesampingkan alternatif pemecahan masalah
3. Bisanya kurang *fleksible* dalam mengahadapi perubahan
4. *Prototype* yang dihasilkan tidak selamanya mudah dirubah
5. *Prototype* terlalu cepat selesai.

Berikut adalah tahap-tahap pembuatan prototype :



Gambar 3.1. Tahap-tahap *prototype*

1. Mengidentifikasikan kebutuhan *user* agar bisa merancang sistem yang akan di bangun sesuai dengan harapan user . Sebelum pada tahap perancangan, tahap awal yang harus dilakukan adalah menganalisis sistem denga cara melakukan pengumpulan data, obervasi, dan wawancara serta membaca literatur yang berhubungan dengan sistem yang akan di bangun, baik dalam model *interface*, teknik, prosedural, maupun teknik yang di gunakan.
2. Pada tahap kedua akan di buat *prototipe* sistem tersebut untuk memperlihatkan kepada pemakai model sistem yang akan di rancang.
3. Pada tahap ketiga yaitu pengujian *prototipe* sistem yang telah di rancang untuk memastikan bahwa sistem tersebut dapat di gunakan dengan baik dan benar sesuai kebutuhan pemakai.

4. Pada tahap keempat yaitu mengembangkan versi produksi, penulis akan merampungkan sesuai dengan masukan terakhir dari pemakai dan memberikan gambaran bagaimana penggunaan sistem tersebut kepada pemakai setelah sistem tersebut di setujui.

3.2.3.3 Alat Bantu Analisis dan Perancangan

Alat Bantu Analisis dan Perancangan yang digunakan dalam penelitian ini yaitu:

1) *Use case* Diagram

Use case atau diagram *use case* merupakan pemodelan untuk kelakukan (*behavior*) sistem informasi yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi itu. *Use case* diagram menggambarkan fungsionalitas dari sebuah system yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. *Use case*

diagram sangat membantu dalam menyusun requirement sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem. Sebuah *use case* dapat meng-include fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-include akan dipanggil setiap kali *use case* yang meng-include dieksekusi secara normal. Sebuah *use case* dapat di-include oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang common. Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain. Adapun syarat penamaan pada *use case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *use case* yaitu pendefinisian apa yang disebut aktor dan *use case*.

a. Aktor merupakan 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 orang.

b. *Use case* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor.

2) *Activity Diagram*

Activity diagram merupakan diagram yang menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity* diagram juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. Diagram aktivitas menunjukkan suatu alur kegiatan secara berurutan dan digunakan untuk mendeskripsikan kegiatan-kegiatan dalam sebuah operasi meskipun juga dapat digunakan untuk mendeskripsikan alur kegiatan yang lainnya seperti *use case* atau suatu interaksi. Diagram aktivitas atau *activity diagram* menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis. Yang perlu diperhatikan disini adalah bahwa diagram aktivitas menggambarkan aktivitas sistem bukan apa yang dilakukan *actor*. Diagram aktivitas juga banyak digunakan untuk mendefinisikan rancangan proses bisnis dimana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan urutan atau pengelompokan tampilan dari sistem / *user interface* dimana setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan rancangan

pengujian dimana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya Diagram aktivitas mendeskripsikan aliran kerja dari perilaku system dan hampir sama dengan diagram status karena kegiatannya merupakan status suatu pekerjaan dengan menunjukkan kegiatan yang dilakukan secara berurutan Sebaiknya diagram aktivitas digunakan untuk melengkapi diagram lain seperti diagram interaksi dan diagram status, karena diagram aktivitas dapat mengetahui aliran sistem yang akan dirancang dan bermanfaat untuk menganalisis *use case* melalui penggambaran aksi-aksi yang dibutuhkan, penggambaran algoritma berurutan yang kompleks, dan pemodelan aplikasi dengan proses parallel Tetapi diagram aktivitas tidak menunjukkan bagaimana objek berperilaku atau objek berkolaborari secara detil.

3) *Sequence Diagram*

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, display, dan sebagainya) berupa *class* yang digambarkan terhadap waktu. Sequence diagram terdiri atas dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). *Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah event untuk menghasilkan output tertentu. Diawali dari apa yang men- *trigger* aktivitas tersebut, proses dan perubahan apa saja

yang terjadi secara internal dan output apa yang dihasilkan. Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal. *Class* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *class* akan dipetakan menjadi operasi/metoda dari *class*. Activation bar menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah *class*.Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan icon khusus untuk objek boundary, controller dan persistent entity. Banyaknya diagram sequence yang harus digambar adalah sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram sequence sehingga semakin banyak *use case* yang didefinisikan maka diagram sequence yang harus dibuat juga semakin banyak.

4) *Collaboration Diagram*

Collaboration diagram menggambarkan interaksi antar objek sama seperti sequence diagram, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian *class*. Setiap *class* memiliki sequence number, di mana *class* dari level tertinggi memiliki nomor 1. *Class* dari level yang sama memiliki prefiks yang sama. Diagram kolaborasi mengelompokkan *class* pada kumpulan diagram sekuen menjadi sebuah diagram. Dalam diagram kolaborasi yang dituliskan

adalah operasi/metode yang dijalankan antara objek yang satu dan objek lainnya secara keseluruhan, oleh karena itu dapat diambil dari jalannya interaksi pada semua diagram sekuen. Penomororan metode dapat dilakukan berdasarkan urutan dijalankannnya metode/operasi diantara objek yang satu dengan objek lainnya atau objek itu sendiri.

5) *Class diagram*

Class diagram adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). *Class diagram* menggambarkan struktur dan deskripsi *class*, package dan objek beserta belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. *Use case* diagram dapat sangat membantu bila ketika menyusun requirement sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang test case untuk semua feature yang ada pada sistem. Sebuah *use case* dapat meng-include fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-include akan dipanggil

setiap kali *use case* yang meng-*include* dieksekusi secara normal. Sebuah *use case* dapat di-include oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang common. Sebuah *use case* juga dapat meng-*extend use case* lain dengan behaviour-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain. *Class diagram* menggambarkan struktur dan deskripsi *class*, package dan objek beserta hubungan antar *class* diantaranya :

a) Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah navigability menunjukkan arah query antar *class*.
b) Agregasi, yaitu hubungan yang menyatakan bagian.
c) Pewarisan, yaitu hubungan hirarkis antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.
d) Hubungan dinamis, yaitu rangkaian pesan (*class*) yang di-passing dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence diagram* yang akan dijelaskan kemudian.

6) *Component Diagram*

Component Diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency)* di antaranya. Komponen piranti lunak adalah modul berisi *code*, baik berisi *source code* maupun binary *code*, baik *library* maupun *executable*, baik yang muncul pada *compile time*, *link time*, maupun *run time*. Umumnya komponen terbentuk dari beberapa *class* dan/atau *package*, tapi dapat juga dari komponen-komponen yang lebih kecil. Komponen dapat juga berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain.

7) *Deployment Diagram*

Deployment/physical diagram menggambarkan detail bagaimana komponen di-*deploy* dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, server atau piranti keras apa), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi server, dan hal-hal lain yang bersifat fisikal . Sebuah node adalah server, *workstation*, atau piranti keras lain yang digunakan untuk men-*deploy* komponen dalam lingkungan sebenarnya. Hubungan antar node (misalnya TCP/IP) dan requirement dapat juga didefinisikan dalam diagram ini.