BAB II
TINJAUAN PUSTAKA

2.1. Kajian Pustaka

Penelitian yang akan dibahas berjudul "*Gap between knowledge and practice in nursing*". Pada penelitian tersebut dibahas tentang adanya ketidakseimbangan antara penerapan teori dan praktik yang disebabkan oleh kurangnya kemampuan teori atau praktik yang dimiliki oleh perawat. Untuk menyelesaikan masalah itu, penelitian tersebut mengkaji dan menyimpulkan upaya untuk menyelesaikan masalah, yaitu memperbaiki cara penyampaian materi oleh pengajar, melakukan pelatihan dimana pelatih adalah perawat yang sudah berpengalaman di dunia kerja, dan memperbaiki cara pembelajaran dimana pendidikan perawat lebih difokuskan pada penerapan teori pada praktik, tidak hanya fokus pada teori [KHS-11]. Tetapi, dengan merealisasikan upaya tersebut dirasa masih kurang dapat menyelesaikan masalah karena mengubah proses pembelajaran yang sudah ada tidaklah mudah dan apa yang dihasilkan dari proses pelatihan perawat belum tentu menjamin bahwa perawat dapat berdaptasi dengan baik pada lingkungan kerja.

Penelitian yang akan dibahas berjudul "*Addressing medication errors – The role of undergraduate nurse education*". Pada penelitian tersebut dibahas bahwa terdapat banyak kesalahan pengobatan yang terjadi. Kesalahan tersebut dapat terjadi ketika proses penulisan resep, penyaluran obat, dan pemberian obat. Kesalahan pada penulisan resep dapat terjadi ketika proses pengambilan keputusan atau penulisan resep itu sendiri. Hal tersebut disebabkan oleh kekurangan pengetahuan tentang obat, pasien, atau keduanya. Selain itu, penulisan resep yang kurang jelas, seperti tidak adanya keterangan alergi terhadap obat, penulisan dosis yang kurang jelas, dan penulisan resep yang sulit dibaca menjadi sumber dari kesalahan penulisan resep. Kesalahan penyaluran obat terjadi karena pasien memiliki nama yang mirip, kesalahan membaca resep, kesalahan penulisan resep turunan dan kurangnya pengalaman tim kesehatan. Sedangkan kesalahan pemberian obat dapat terjadi karena kecerobohan, kelalaian, dan kurang perhatian

pada saat pemberian obat. Selain itu, kesalahan pemberian obat bisa juga terjadi karena kurangnya pengetahuan tentang obat tersebut [KPA-07].

Berdasarkan penjelasan diatas, terjadinya kesalahan pengobatan tidak hanya disebabkan oleh kesalahan perawat, tetapi dapat disebabkan oleh tim kesehatan lain, seperti dokter, apoteker, dan lain-lain. Keterlibatan perawat pada proses pemberian obat, membuat perawat harus mampu mempertimbangkan bahwa ketika perawat mengelola obat-obatan, perawat menjadi bertanggung jawab tidak hanya untuk kesalahan perawat sendiri tetapi juga untuk mengakui bahwa karena berbagai faktor kesalahan mungkin terjadi ketika penentuan dan pembagian obat. Oleh karena itu, secara tidak langsung perawat harus bisa mengidentifikasi dan mencegah terjadinya kesalahan pemberian obat. Untuk itu, perawat harus dilengkapi dengan pengetahuan tentang obat-obatan [KPA-07].

Penelitian yang akan dibahas berjudul "*A closer look at nursing documentation on paper forms: Preparation for computerizing a nursing documentation system*". Pada penelitian tersebut dibahas tentang pentingnya catatan perawat yang berisikan tentang keadaan kesehatan pasien serta catatan tindakan medis yang dilakukan oleh perawat atau dokter yang menangani pasien tersebut. Pada penelitian tersebut dijelaskan bahwa pentingnya sebuah catatan perawat yang berkaitan dengan apakah tindakan penanganan yang dilakukan oleh perawat sudah benar atau tidak, serta tindakan perawat apakah sesuai dengan tujuan perawatan pasien. Selain itu, penelitian tersebut juga mencoba untuk membuat aplikasi untuk membuat catatan dengan format tertentu yang diterapkan pada komputer. Kelebihan yang ditawarkan aplikasi tersebut, yaitu kinerja perawat akan sesuai dengan tujuan perawatan pasien, tetapi terdapat kerugian yaitu format catatan perawat tidak digunakan pada semua rumah sakit dan aplikasi diterapkan pada komputer yang dapat mengganggu kinerja perawat yang membutuhkan mobilitas tinggi [HKP-10].

2.2. Buku Saku Perawat

Buku saku adalah buku berukuran kecil yang dapat dimasukkan ke dalam saku dan mudah dibawa kemana-mana [PBN-14]. Berdasarkan pengertian buku

saku tersebut, maka dapat disimpulkan bahwa buku saku perawat adalah buku saku yang berisikan materi tentang keperawatan.

2.3. JSON

JSON (JavaScript Object Notation) adalah format pertukaran data yang ringan, mudah dibaca, dan ditulis oleh manusia. Bisa juga diartikan dengan format sederhana untuk melakukan pertukaran data antara *browser* dan *server* [KSI-08]. Selain itu, data berformat JSON mudah diuraikan dan dibuat oleh komputer. JSON merupakan bagian dari Bahasa Pemrograman JavaScript, Standar ECMA-262 Edisi ke-3 Desember 1999. JSON merupakan format teks berbahasa independen, tetapi menggunakan konversi yang akrab bagi *programmer* dari keluarga besar bahasa C, seperti C, C++, C#, Java, JavaScript, Perl, Python, dan sebagainya. Hal tersebut membuat JSON menjadi bahasa pertukaran data yang ideal [JSN-14].

JSON dibentuk dari 2 struktur, yaitu [KSI-08]:

1. Kumpulan pasangan nama/nilai. Dalam berbagai bahasa, hal ini direalisasikan sebagai sebuah objek, rekaman, struktur, kamus, tabel hash, daftar berkunci, atau array asosiatif.
2. Daftar nilai berurutan. Pada kebanyakan bahasa, hal ini direalisasikan sebagai sebuah array, vektor, atau urutan.

Struktur data JSON adalah struktur data universal. Pada dasarnya, semua bahasa pemrograman modern mendukung struktur data tersebut dalam bentuk yang sama maupun berlainan. Hal tersebut dapat dikatakan demikian karena format data mudah dipertukarkan dengan bahasa pemrograman yang berdasarkan pada struktur data tersebut [JSN-14].

2.3.1. Tipe Data JSON

Pada JSON, data dapat berupa string, angka, objek, atau larik. String adalah kumpulan dari nol atau lebih karakter Unicode atau *backslash escapes* "\" dalam petik ganda. String biasanya merupakan string dari bahasa C atau Java. Berikut ini merupakan tipe data JSON [KSI-08]:

1. *Number*, dapat berupa angka, integer, atau float.

2. String, dapat berupa teks atau karakter Unicode dalam petik ganda dengan *backslash escapes*.
3. Boolean, tipe data Boolean merepresentasikan nilai benar atau salah.
4. Null, nilai null merepresentasikan bahwa variabel tidak memiliki nilai.
5. *Object*, kumpulan pasangan nilai yang dipisahkan dengan koma "," dan dimulai dengan kurung kurawal buka "{", serta diakhiri dengan kurung kurawal tutup "}".
6. *Array*, urutan nilai yang berurutan yang dipisahkan dengan "," dan dimulai dengan kurung siku buka "[", serta diakhiri dengan kurug siku tutup "]".

Pada bentuk inti, JSON merepresentasikan 4 tipe data primitif seperti string, angka, Boolean, null, dan 2 tipe data terstruktur, yaitu objek dan larik. Tipe data tersebut menggunakan kode *escape* JavaScript standard dan menambahkan sebuah *backslah* "\" sebelum menuliskan karakter berikut [KSI-08]:

1. " (tanda petik)
2. b (*backspace*)
3. n (baris baru)
4. f (*form feed*)
5. r (*carriage return*)
6. t (tab horisontal)
7. u (ditambah 4 digit untuk sebuah karakter Unicode)
8. \ (*backslash*)
9. / (*forward slash*)

2.3.2. Contoh Data JSON

Berikut ini merupakan contoh data berfomat JSON yang akan dijelaskan pada Gambar 2.1.

```
{
    "Details":
    {
        "FirstName": "Suchita",
        "LastName": "Jain",
        "Address":
        {
```

```
            "StreetAddress": "43 4th Street",
            "Country": "India",
            "State": "New Delhi",
            "PostalCode": 110002
        },
        "PhoneNumber": 0112345689
    }
}
```

Gambar 2.1. Contoh Data JSON

Sumber: [KSI-08]

Gambar 2.1. menjelaskan bahwa `Details` adalah objek tertinggi dan semua data lainnya adalah anggota dari objel tersebut. `FirstName`, `LastName`, dan `Address` adalah kolom string dan `PostalCode` dan `PhoneNumber` adalah kolom angka. Pasangan nama dan nilai dipisahkan dengan koma, misalnya `"FirstName": "Suchita"` adalah salah satu pasangan nama/nilai dan pasangan nama/nilai lainnya yaitu, `"LastName": "Jain"`, maka koma akan memisahkan 2 pasangan tersebut. Selain itu, titik dua ":" digunakan antara nama dan nilai [KSI-08].

2.4. *Fragment*

Fragment merupakan kunci untuk mengimplementasikan antarmuka yang responsive pada perangkat Android. Sebuah *fragment* adalah sebuah bagian antarmuka independen yang dapat ditambahkan ke sebuah *layout*. Cara mengimplementasikan *fragment* sangat mirip dengan cara membuat *activity*, yaitu dengan membuat implementasi *class* dan menentukan *layout* untuk menggambar antarmuka *fragment* [JHL-12].

2.4.1. Membuat Fragment

Untuk membuat *fragment* dilakukan dengan meng*extend* `Fragment` *class* atau salah satu *subclass* `DialogFragment`, `ListFragment`, `PreferenceFragment`, atau `WebViewFragment`.

Kebanyakan pengetahuan tentang bekerja dengan *activity* dapat diterapkan untuk bekerja dengan *fragment*, dengan beberapa pengecualian. Pertama, banyak *method* yang tidak disediakan oleh *fragment superclass*, dan *fragment* bukan sebuah objek `Context`. Untuk memperoleh objek `Context`, panggil *method* `getActivity()`. *Method* tersebut akan mengembalikan *activity* yang terdapat *fragment* yang terlampir didalamnya. Cara tersebut dapat digunakan untuk memanggil objek `Context`. Tetapi, *method* tersebut dapat mengembalikan nilai `null` jika *fragment* tidak dilampirkan pada *activity*.

Kedua, *fragment* tidak dapat digunakan untuk memanggil *method* `setContentView()` untuk menentukan antarmuka *fragment*. Bahkan, antarmuka *fragment* dibuat dengan *method* `onCreateView()`, sehingga harus meng*override method* tersebut untuk memperoleh antarmuka tampilan yang didapatkan dari kembalian *method* tersebut [JHL-12].

2.4.2. Siklus Hidup *Fragment*

Fragment memiliki *method* siklus hidup yang mirip dengan *activity*. *Method* tersebut dapat digunakan untuk menghentikan dan memulai fungsionalitas dengan cara yang sama seperti pada *activity* [JHL-12]. Untuk mengimplementasikan *fragment* biasanya digunakan *method* siklus hidup berikut [ADV-14]:

1. onCreate(). Sistem akan memanggil *method* ini kekita membuat *fragment*. Dalam mengimplementasikannya, harus menginisialisasi komponen penting dari *fragment* yang ingin dipertahankan ketika fragment ditunda atau dihentikan, kemudian dilanjutkan.
2. onCreateView(). Sistem akan memanggil *method* ini kekita saatnya *fragment* mengambil antarmuka untuk pertama kalinya. Untuk menggambar antarmuka *fragment*, *method* harus mengembalikan tampilan yang diambil dari *layout fragment*. *Method* dapat mengembalikan nilai null jika *fragment* tidak menyediakan antarmuka.
3. onPause(). Sistem akan memanggil *method* ini sebagai indikasi pertama bahwa pengguna meninggalkan *fragment* (meskipun tidak selalu berarti *fragment*

sedang dihancurkan). Hal ini biasanya terjadi ketika harus melakukan perubahan yang harus dipertahankan diluar *session* pengguna saat ini (karena pengguna mungkin tidak akan kembali).

Sebagian besar aplikasi harus menerapkan setidaknya tiga *method* untuk setiap fragmen, tetapi ada beberapa *callback method* yang dapat digunakan untuk menangani berbagai tahap siklus hidup *fragment* [ADV-14].

2.4.3. Menambahkan *Fragment* ke *Layout*

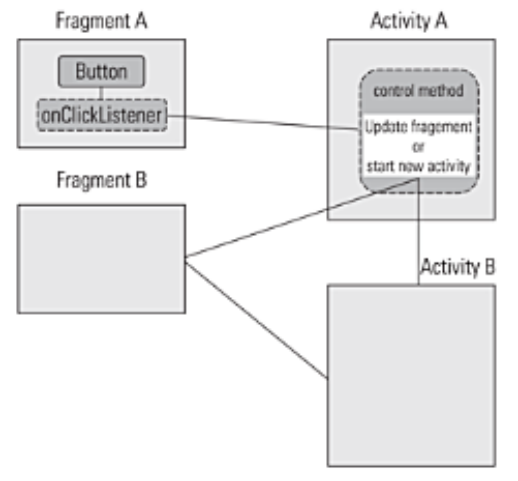
Fragment saja tidak terlalu berguna. Oleh karena itu, harus menambahkan *fragment* ke *layout*. Ada 2 cara untuk melakukan hal tersebut, yaitu menggunakan elemen XML pada dokumen *layout* dan menambahkan fragment secara dinamis dengan kode menggunakan `FragmentManager` *class* [ADV-14].

2.4.4. Arsitektur *Fragment* dan *Activity*

Cara membangun interaksi antara *activity* dan *fragment* mendeskripsikan bagaimana antarmuka bisa fleksibel. *Fragment* adalah modul antarmuka, tetapi *fragment* tersebut juga harus modular. Makin sedikit *fragment* yang mengetahui *fragment* lain dan bahkan tentang *parent activity* dari *fragment* tersebut, maka *fragment* makin fleksibel dan dapat digunakan pada konfigurasi antarmuka berbeda.

Aturan paling penting yang harus diikuti adalah *fragment* tidak boleh berkomunikasi secara langsung dengan *fragment* lainnya. Semua komunikasi harus dilakukan melalui *activity*. Jika *fragment* harus berkomunikasi secara langsung, maka sistem akan dipaksa untuk selalu menampilkan *fragment* tersebut pada waktu yang bersamaan.

Activity akan mengontrol alur aplikasi sebelum *fragment* muncul. Ketika sebuah control pada *fragment* membutuhkan perubahan pada bagian lain antarmuka, maka *fragment* akan memanggil *parent activity* [JHL-12]. Berikut ini merupakan contoh dari alur kontrol aplikasi yang akan dijelaskan pada Gambar 2.2.



Gambar 2.2. Contoh Alur Kontrol Aplikasi

Sumber: [JHL-12]

Gambar 2.2. menjelaskan ketika pengguna menekan tombol pada *fragment* A yang akan mempengaruhi antarmuka di dalam *fragment* itu sendiri. Control dikirim ke *parent activity*, yang akan ditentukan berdasarkan pada *layout* saat ini apakah sebuah *activity* baru ingin diluncurkan atau *fragment* yang sudah ada ingin diubah atau diperbarui. Struktur tersebut menjamin bahwa *fragment* A dan B dapat digunakan pada *activity* yang sama atau pada *activity* berbeda [JHL-12].

2.5. *Software Process Model*

Sebuah model proses perangkat lunak adalah deskripsi yang disederhanakan dari proses perangkat lunak yang disajikan menjadi satu tampilan dari proses tersebut. Proses model dapat mencakup kegiatan yang merupakan bagian dari proses perangkat lunak, produk perangkat lunak, dan peran dari orang yang terlibat dalam rekayasa perangkat lunak. Berikut ini merupakan beberapa contoh jenis model proses perangkat lunak [ISO-07]:

1. *Workflow model*. Model ini menunjukkan urutan kegiatan dalam proses bersamaan dengan masukan, keluaran, dan ketergantungan. Kegiatan dalam model ini merepresentasikan tindakan manusia.
2. *Dataflow* atau *activity model*. Model ini merepresentasikan proses sebagai serangkaian kegiatan, yang masing-masing melakukan beberapa transformasi

data. Hal ini menunjukkan bagaimana input ke proses, seperti spesifikasi, ditransformasikan ke output, seperti desain. Kegiatan ini merepresentasikan transformasi yang dilakukan oleh orang-orang atau komputer.

3. *Role/Action model*. Model ini merepresentasikan peran dari orang-orang yang terlibat dalam proses perangkat lunak dan kegiatan yang menjadi tanggung jawabnya.

Kebanyakan model proses perangkat lunak didasarkan pada salah satu dari tiga model umum atau proses paradigma pengembangan perangkat lunak berikut [ISO-07]:

1. *Waterfall model*. Pendekatan ini mengambil kegiatan proses dasar dari spesifikasi, pengembangan, validasi, dan evolusi perangkat lunak, serta mewakili proses tersebut sebagai tahap proses terpisah seperti spesifikasi kebutuhan, perancangan perangkat lunak, implementasi, pengujian dan seterusnya. Setelah setiap tahap didefinisikan itu adalah *sign-off* dan pembuatan perangkat lunak akan dilanjutkan ke tahap berikutnya.

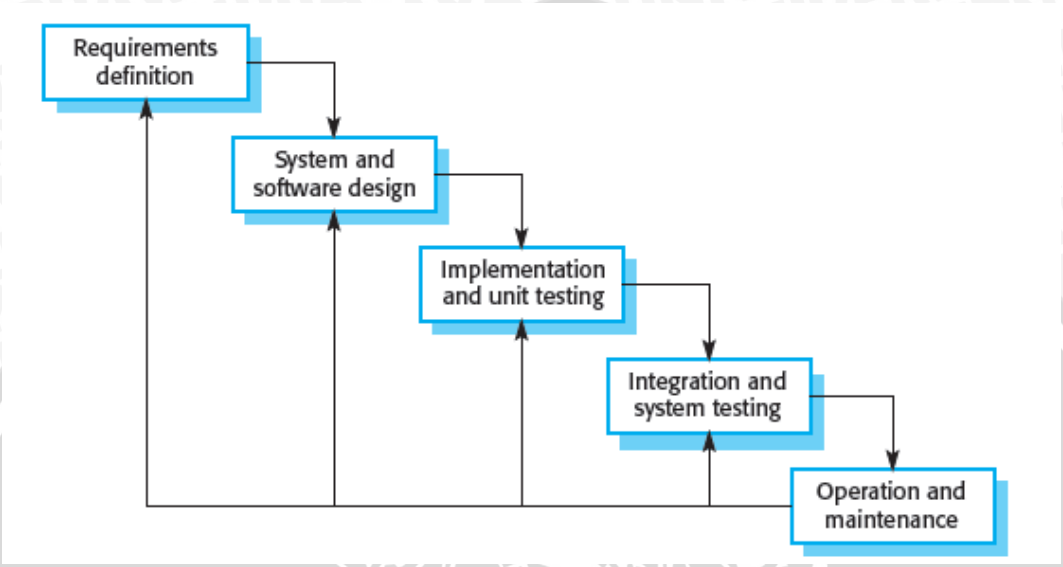
2. *Evolutionary development*. Pendekatan ini memisahkan kegiatan dari spesifikasi, pengembangan, dan validasi. Sistem awal dengan cepat dikembangkan dari spesifikasi abstrak. Hal ini kemudian disempurnakan dengan masukan pelanggan untuk menghasilkan sistem yang memenuhi kebutuhan pelanggan.

3. *Component-based software engineering* (CBSE). Pendekatan ini berdasarkan pada eksistensi dari sejumlah komponen yang dapat digunakan lagi. Proses pengembangan sistem berfokus pada mengintegrasikan komponen tersebut ke sistem daripada mengembangkan sistem dari awal.

Ketiga model tersebut saat ini banyak digunakan dalam rekayasa perangkat lunak. Ketiga model tersebut tidak saling eksklusif dan sering digunakan bersama, terutama untuk pengembangan sistem yang besar. Sub-sistem dalam sistem yang lebih besar dapat dikembangkan dengan menggunakan pendekatan yang berbeda. Oleh karena itu, meskipun akan lebih mudah untuk membahas model ini secara terpisah, harus memahami bahwa dalam praktik, ketiga model tersebut sering digabungkan [ISO-07].

Dalam pembuatan perangkat lunak, aplikasi buku saku perawat akan dikembangkan menggunakan model *waterfall*. Model *waterfall* adalah model

yang diterbitkan pertama dari proses pengembangan perangkat lunak berasal dari proses rekayasa sistem yang lebih umum [ISO-07]. Hal ini digambarkan pada Gambar 2.3.



Gambar 2.3. Model Pengembangan Perangkat Lunak *Waterfall*

Sumber: [ISO-07]

Berikut ini merupakan tahapan dari model yang digambarkan pada Gambar 2.3 [ISO-07].

1. *Requirement analysis and definition*. Layanan, kendala dan tujuan sistem ditetapkan melalui konsultasi dengan pengguna sistem, kemudian didefinisikan secara rinci yang nantinya berfungsi sebagai spesifikasi sistem.
2. *System software and design*. Proses desain sistem membagi kebutuhan baik perangkat keras atau perangkat lunak sistem. Hal ini ditapkan pada keseluruhan arsitektur sistem. Desain perangkat lunak melibatkan identifikasi dan gambaran dasar abstraksi sistem perangkat lunak dan hubungannya.
3. *Implementation and unit testing*. Selama tahap ini, desain perangkat lunak diwujudkan sebagai satu kumpulan program atau unit program. Pengujian unit melibatkan verifikasi bahwa setiap unit memenuhi spesifikasinya.
4. *Integration and system testing*. Unit program atau program individu diintegrasikan dan diuji sebagai sistem yang lengkap untuk memastikan bahwa

kebutuhan perangkat lunak telah dipenuhi. Setelah pengujian, sistem perangkat lunak disampaikan kepada pelanggan.

5. *Operation and maintenance*. Biasanya (meskipun tidak selalu dibutuhkan) merupakan tahap terpanjang dalam tahap siklus hidup perangkat lunak. Sistem ini dipasang dan diterapkan dalam penggunaan praktis. Pemeliharaan melibatkan pengecekan kesalahan yang tidak ditemukan pada tahap awal siklus hidup, meningkatkan implementasi unit sistem, dan meningkatkan layanan sistem sebagai kebutuhan yang baru ditemukan.

Pada prinsipnya, hasil dari setiap tahap adalah satu atau lebih dokumen yang disetujui ('ditandatangani'). Tahap berikut ini tidak boleh dimulai sampai tahap sebelumnya selesai. Dalam prakteknya, tahap ini tumpang tindih dan memberikan informasi satu sama lain. Misalnya, selama proses desain, masalah dengan kebutuhan diidentifikasi.

Model proses perangkat lunak bukan model linier sederhana namun melibatkan urutan iterasi dari kegiatan pembangunan. Oleh karena itu, untuk menghemat biaya, masalah yang tersisa untuk resolusi kemudian diabaikan. Pembekuan kebutuhan prematur mungkin berarti bahwa sistem tidak akan melakukan apa yang diinginkan pengguna [ISO-07].

Selama fase siklus hidup akhir (operasi dan pemeliharaan), perangkat lunak mulai digunakan. Kesalahan dan kelalaian dalam kebutuhan perangkat lunak yang sebenarnya ditemukan. Program dan desain kesalahan muncul dan kebutuhan untuk fungsi baru diidentifikasi. Oleh karena itu, sistem harus berevolusi untuk tetap berguna.

Keuntungan dari model *waterfall* adalah bahwa dokumentasi diproduksi pada setiap tahap dan bahwa itu cocok dengan model proses rekayasa lainnya. Masalah utamanya adalah pembagian proyek menjadi tahap yang berbeda tidak fleksibel. Komitmen harus dibuat pada tahap awal dalam proses, yang membuatnya sulit untuk merespon perubahan kebutuhan pelanggan.

Oleh karena itu, model *waterfall* harus digunakan hanya ketika kebutuhan dipahami dengan baik dan tidak terjadi perubahan secara radikal selama pengembangan sistem. Namun, model *waterfall* mencerminkan jenis model proses yang digunakan dalam rekayasa perangkat lunak lainnya. Akibatnya, proses

perangkat lunak berbasis pada pendekatan ini masih digunakan untuk mengembangan perangkat lunak, terutama ketika proyek perangkat lunak merupakan bagian dari proyek rekayasa sistem yang lebih besar [ISO-07].