BAB II

TINJAUAN PUSTAKA

2.1 Tanaman Herbal

Tanaman obat adalah tanaman yang memiliki khasiat obat dan digunakan sebagai obat dalam penyembuhan maupun pencegahan penyakit. Pengertian berkhasiat obat adalah mengandung zat aktif yang berfungsi mengobati penyakit tertentu atau jika tidak mengandung zat aktif tertentu tapi mengandung efek resultan/sinergi dari berbagai zat yang berfungsi mengobati (Flora, 2008). Herbal sering disebut sebagai tanaman obat, sehingga dalam pengembangannya dimasukkan sebagai salah satu bentuk pengobatan alternatif. Dari besarnya jumlah jenis tanaman yang ada di Indonesia, yaitu sekitar 30.000 jenis (Kominfonewscenter, 2010), telah terdapat 261 jenis tanaman yang telah dikenal dan memiliki kegunaan dalam pengobatan.

2.2 Aplikasi *smartphone*

Aplikasi merupakan penerapan dari rancang sistem untuk mengolah data menggunakan aturan atau ketentuan bahasa pemrograman tertentu (Kamus Besar Bahasa Indonesia, 1992). *Smartphone* merupakan telepon selular dengan mikro prosesor, memori, layar dan modem bawaan (Williams & Sawyer, 2011). *Smartphone* merupakan ponsel multimedia yang menggabungkan fungsionalitas PC dan *handset* sehingga menghasilkan *gadget* yang mewah. *Smartphone* memiliki fitu pesan teks, kamera, pemutar musik, *video*, *game*, akses *email*, tv *digital*, *search*

engine, pengelola informasi pribadi, fitur *GPS*, dan jasa telepon internet, bahkan terdapat telepon yang juga berfungsi sebagai kartu kredit. Jadi aplikasi *smartphone* adalah sebuah penerapan dari rancang sistem untuk mengolah data menggunakan aturan atau ketentuan dari bahasa pemrograman tertentu yang digunakan pada sebuah *smartphone*.

2.3 *Ontology*

Ontology memiliki definisi formal yaitu suatu bahasa dengan komitmen ontologis, sehingga ontologi terhadap bahasa adalah kumpulan aksioma yang dirancang sedemikian rupa sehingga didapat himpunan model yang diperkirakan akan menghasilkan model terbaik sesuai dengan komitmen (Guarino, 1998).

Guarino juga memperkenalkan konsep Sistem Informasi (SI) berbasis ontologi. Ontologi menggerakkan bagian – bagian dari SI, yaitu program aplikasi, sumber daya informasi, dan antarmuka pemakai. Dampak ontologi dalam SI dapat dilihat dalam 2 sisi, yaitu

1. Dimensi temporal, yaitu dampak ontologi saat pengembangan dalam SI atau saat pelaksanaan.
2. Dimensi struktural, yaitu dampak ontologi pada komponen – komponen SI, seperti basisdata, antarmuka pemakai, dan program aplikasi.

Selain itu obyek – obyek yang terlibat dalam keterkaitan antar mereka tergambar dalam perbedaharaan kata dan diolah oleh program sehingga menghasilkan pengetahuan (Gruber, 1995). Penentuan penampilan memerlukan rancangan keputusan berdasarkan kriteria obyektif sesuai hasil yang diinginkan.

Berikut kriteria perancangan dengan tujuan berbagai pengetahuan dan kerjasama antar program:

1. Kejelasan, yaitu suatu ontologi dapat menjelaskan arti yang diinginkan secara obyektif.
2. Keterkaitan, yaitu suatu ontologi harus saling terkait antar obyek–obyeknya dan konsisten sesuai definisi.
3. Dapat diperluas, yaitu ontologi dirancang untuk mengantisipasi perbendaharaan kata yang saling berbagi.
4. Bias pengkodean minimal, yaitu penggagasan harus ditentukan pada tingkat pengetahuan tanpa bergantung pada pengkodean dengan simbol–simbol tertentu.
5. Komitmen ontologis minimal, yaitu suatu ontologi membutuhkan komitmen ontologis secara minimal dalam mendukung kegiatan berbagi pengetahuan.

Pengkodean secara intrinsik terhadap konsep dapat menghindari bias terhadap arti, yaitu dengan menyatakan konsep yang mendasar (contoh *time points*) dan satuan pengukuran (contoh *time point years*). Hasil dari perincian terhadap batasan-batasan yang ada membuat program dan pengetahuan menghasilkan kesimpulan yang berguna.

2.5 *Web Ontology Language* (OWL)

Web Ontology Language atau yang biasa disebut *OWL* merupakan sebuah bahasa yang digunakan oleh aplikasi–aplikasi dan tidak hanya menampilkan informasi tersebut pada manusia, melainkan juga yang perlu memproses isi informasi tersebut. *OWL* juga merupakan *extension* dari *RDF data model* untuk menyediakan *semantic* yang sangat kaya, untuk membuat sebuah *data models* yang kompleks (W3C, 2004).

OWL memiliki tiga buah sub bahasa sebagai berikut.

1. *OWL Lite*

   Aslinya *OWL Lite* diperuntukkan sebagai *support* untuk *user* yang memerlukan klasifikasi hierarki dan *simple constraint* sebagai yang utama.

2. *OWL DL*

   OWL DL dirancang untuk menyediakan kecepatan semaksimal mungkin sambil mempertahankan kesempurnaan komputasi, *decidability*, dan ketersediaan dari logika algoritma praktikal.

3. *OWL Full*

   OWL Full didasarkan oleh *semantics* yang berbeda dengan *OWL Lite* atau *OWL DL* dan di *design* untuk menyimpang beberapa kecocokan dengan *RDF schema.*

2.6 *Java Programming Language*

Java Programming Language adalah bahasa pemrograman yang dapat dijalankan di berbagai komputer termasuk telepon genggam. Bahasa ini awalnya dibuat oleh James Gosling saat masih bergabung di *Sun Microsystems* saat ini merupakan bagian dari *Oracle* dan dirilis tahun 1995. Bahasa ini banyak mengadopsi sintaksis yang terdapat pada C dan C++ namun dengan sintaksis model obyek yang lebih sederhana serta dukungan rutin-rutin aras bawah yang minimal. Aplikasi-aplikasi berbasis java umumnya dikompilasi ke dalam *p-code (bytecode)* dan dapat dijalankan pada berbagai Mesin Virtual Java / *Java Virtual Machine (JVM)* (Computerweekly, 2002).

Java merupakan bahasa pemrograman yang bersifat umum atau non-spesifik dan secara khusus dirancang untuk memanfaatkan dependensi implementasi seminimal mungkin. Fungsionalitasnya memungkinkan aplikasi *Java* mampu berjalan dibeberapa *platform* sistem operasi yang berbeda, *Java* dikenal pula dengan slogannya, "Tulis sekali, jalankan di mana pun". *Java* memiliki kelebihan dibanding bahasa pemrograman lain, yaitu:

1. *Multiplatform*, yaitu *Java* dapat dijalankan pada beberapa *paltform* atau sistem operasi komputer yang berbeda, sesuai dengan slogannya.
2. *OOP* atau *Object Oriented Programming*
3. Perpustakaan atau *library* yang lengkap, *Java* dikenal dengan kumpulan *library* yang sangan luas, ditambah dengan keberadaan komunitas *Java*

yang terus membuat *library* baru untuk melingkupi seluruh kebutuhan pengembangan aplikasi.

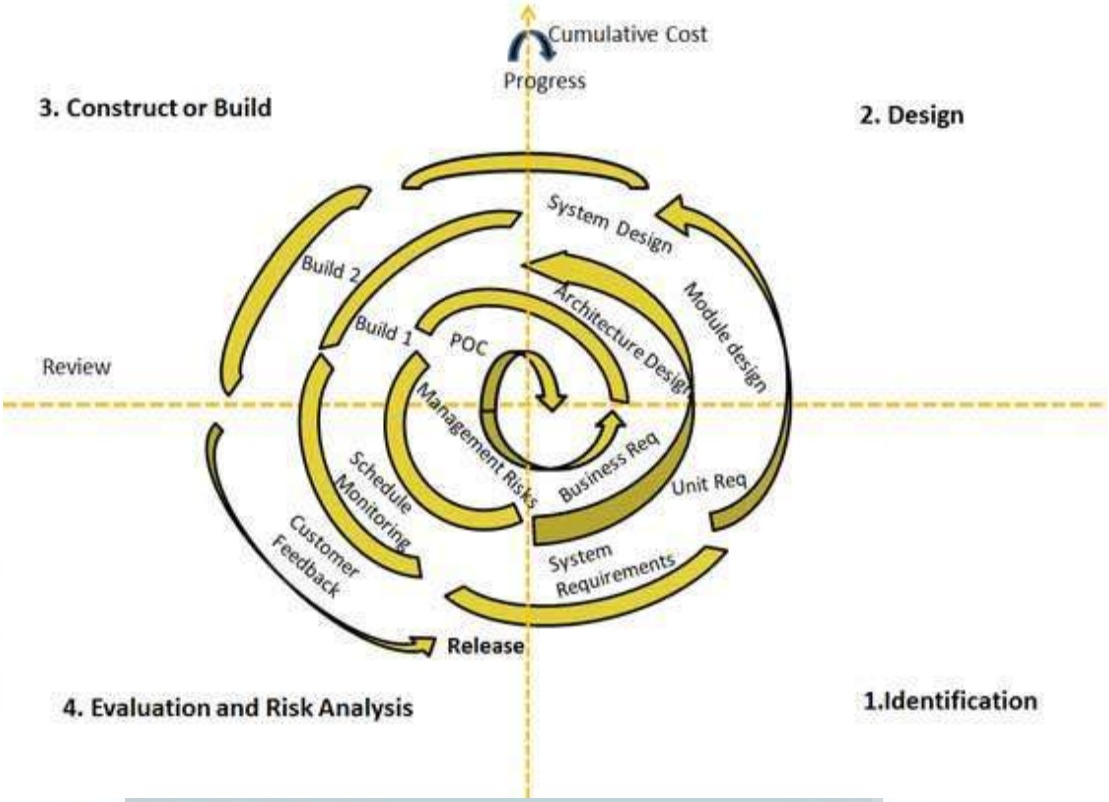
4. Bergaya C++ shingga menarik banyak pemrogram C++ untuk pindah ke *Java.*
5. Pengumpulan sampah atau *garbage collection* secara otomatis, sehingga pemrogram tidak perlu melakukan pengaturan memori secara langsung.

2.7 Android Operating System

Sistem operasi *Android* adalah sebuah sistem operasi berbasis Linux yang dirancang untuk perangkat bergerak layar sentuh seperti *smartphone* dan *tablet computer*. Pada awalnya dikembangkan oleh *Android*, Inc., dengan dukungan finansial dari Google, yang kemudian dibeli pada tahun 2005. Sistem operasi ini resmi dirilis pada tahun 2007 bersamaan dengan didirikannya *Open Handset Alliance,* konsorsium dari perusahaan-perusahaan perangkat keras, perangkat lunak, dan telekomunikasi yang bertujuan untuk memajukan standar terbuka perangkat seluler (Openhandsetalliance, 2007).

2.8 SDLC Spiral Model

Spiral model memadukan dari proses pengembangan *iterative* dan model pengembangan *squential linear* contohnya seperti model pengembangan *waterfall (*SDLC *Waterfall Model)* dengan tekanan yang sangan tinggi kepada penganalisaan resiko. Metode ini memperbolehkan *incremental release* pada produk atau *incremental refinement* melalui masing–masing iterasi mengelilingi lingkaran (Mooz & Forsberg, 2001).



Gambar 2. 1. SDLC Spiral Model

Sumber: (Tutorialspoint, 2017)

SDLC *Spiral* memiliki 4 fase, suatu projek perangkat lunak melewati fase-fase ini secara iterasi yang bernama *spiral.*

1. *Identification*

   Fase dimulai dengan mengumpulkan kebutuhan bisnis dengan dasar *spiral*. *Spiral* berikutnya sejalan dengan pembuatan produk, identifikasi dari kebutuhan sistem dan subsistem, serta kebutuhan unit semua dijalankan pada fase ini. Fase ini juga termasuk memahami kebutuhan sistem dengan komunikasi terus menerus dengan *customer* dan sistem analis.

2. *Design*

Fase *design* dimulai dengan *conceptual design* sebagai dasar, dan melibatkan *architectural design, logical design* dari modul, *physical product design,* dan *final design* pada *spiral* berikutnya.

3. *Construct* atau *Build*

Fase *construct* mengarah kepada membuat perangakat lunak produk yang sebenarnya pada setiap *spiral*. Pada *spiral* dasar atau awal pada saat produk baru saja di pikirkan dan *design* baru saja di buatkan sebuah POC (*Proof of Concept)* untuk mendapatkan masukkan dari pelanggan.

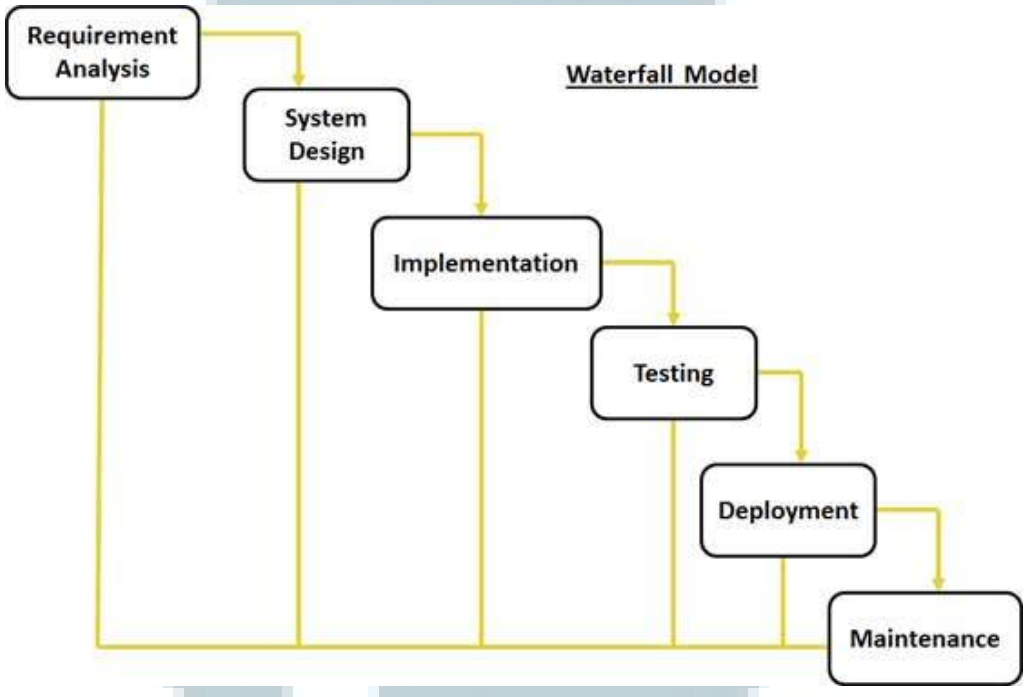
Kemudian pada *spiral* berikutnya dengan kejelasan yang lebih baik pada kebutuhan dan detail *design* yang lebih baik, dibuat model yang sudah berjalan yang disertai dengan menyertakan nomor versi, yang kemudian akan dikirimkan kepada pelanggan dan diminta masukkannya kembali.

4. *Evaluation* dan *Risk Analysis*

Fase analisis resiko termasuk dengan melakukan identifikasi, estimasi, dan memantau kemungkinan *technical* dan resiko manajemen, seperti jadwal tidak sesuai dan habisnya dana. Setelah melakukan *testing,* pada akhir *spiral* pelanggan akan

melakukan evaluasi mengenai perangkat lunaknya, dan memberikan masukan sebelum memasuki *spiral* berikutnya.

2.9 *SDLC Waterfall Model*



Gambar 2. 2. SDLC Waterfall Model

Sumber: (Tutorialspoint, 2017)

SDLC *Waterfall Model* merupakan proses model pertama yang dikenalkan. Proses ini juga dikenal dengan sebutan *liner-sequential life cycle model.* Model ini sangat simple untuk dimengerti dan gunakan. Pada model ini setiap fase harus diselesaikan sebelum dapat masuk atau memulai ke fase berikutnya dan tidak diperbolehkan pengerjaan fase yang tidak terurut (Mooz & Forsberg, 2001). Pada pendekatan *"The Waterfall"* seluruh proses pembuatan perangkat lunak dibagi kedalam beberapa fase, dan dalam model ini biasanya hasil dari suatu fase akan bertindak sebagai masukan atau *input* pada fase berikutnya, berikut adalah fase – fase yang ada pada SDLC *Waterfall Model:*

1. *Requirement Gathering and analysis*

   Semua kemungkinan kebutuhan dari sistem yang akan dibangun akan ditangkap atau ditemukan pada fase ini, dan akan di dokumentasikan kedalam *requirement specification doc.*

2. *System Design*

   Requirement specifications doc dari fase pertama akan dipelajari pada fase ini, dan *system design* akan disiapkan. *System design* akan membantu dalam menentukan kebutuhan *hardware* serta *system* yang akan dibutuhkan, serta membantu menjelaskan arsitektur *system* secara keseluruhan.

3. *Implementation*

   Dengan masukan atau *input* dari *system design*, kemudian pertama – tama *system* akan dibangun dalam sebuah *program* kecil bernama *units,* yang akan kemudian diintegrasikan pada fase berikutnya, setiap *unit* dibangun dan akan di *test* pada fungsinya, yang akan mengacu pada *Unit Testing.*

4. *Integration and Testing*

   Semua *units* yang telah dibangun pada fase implementasi akan diintegrasikan kedalam *system* setelah semua *unit* sudah di *test.* Setelah diintegrasi seluruh sistem akan di *test* kembali untuk mencari kesalahan dan kegagalan yang ada.

5. *Deployment of System*

Setelah semua fungsional dan non fungsional telah selesai di *test*, produk akan dikirimkan ke lingkungan *customer* atau dilepas di pasar.

6. *Maintenance*

   Akan terdapat beberapa masalah pada lingkungan *client*. Untuk memperbaikinya *patches* akan dikeluarkan, atau untuk meningkatkan produk, versi yang lebih baik juga dikeluarkan. *Maintenance* ada untuk mengantarkan perubahan–perubahan tersebut ke lingkungan *customer*.

2.10 *UML (Unified Modeling language)*

Unified Modeling Language adalah keluarga notasi grafis yang didukung oleh meta-model 28 tunggal, yang membantu pendeskripsian dan desain sistem perangkat lunak, khususnya sistem yang dibangun menggunakan pemrograman berorientasi obyek (Fowler, 2005).

Unified Modeling Language atau UML ini memiliki beberapa tujuan, antara lain adalah

1. Dapat memberikan bahasa permodelan visual kepada pengguna dari berbagai macam pemrograman maupun proses rekayasa.
2. Dapat memodelkan sistem yang berkonsep berorientasi obyek, jadi tidak hanya digunakan untuk memodelkan perangkat lunak (*software*) saja.

3. Dapat menciptakan suatu bahasa permodelan yang nantinya dapat dipergunakan oleh manusia maupun oleh mesin.

UML memiliki beberapa jenis diagram yang berbeda–beda, antara lain

1. *Use Case Diagram*

   Merupakan diagram yang menggambarkan interaksi antara sistem dan aktor, *use case diagram* juga mendeskripsikan tipe interaksi antara si pemakai sistem dengan sistemnya.

2. *Activity Diagram*

   Merupakan diagram yang dapat memodelkan proses – proses apa saja yang terjadi pada sistem.

3. *Sequence Diagram*

   Merupakan diagram yang menjelaskan interaksi obyek yang berdasarkan urutan waktu, juga menggambarkan urutan atau tahapan yang harus dilakukan untuk dapat menghasilkan sesuatu seperti pada *use case diagram*.

4. *Class Diagram*

   Merupakan diagram yang digunakan untuk menampilkan kelas–kelas maupun paket–paket yang ada pada suatu sistem yang nantinya akan digunakan.

5. *Statemachine Diagram*

   Merupakan diagram yang menggambarkan transisi maupun perubahan keadaan suatu obyek pada sistem.

6. *Communication Diagram*

Merupakan diagram yang menggambarkan tahapan terjadinya suatu aktivitas dan juga interaksi antara obyek yang ada pada sistem.

7. *Deployment Diagram*

   Merupakan diagram yang menunjukkan tata letak suatu sistem secara fisik, dapatjuga untuk menampilkan bagian – bagian *software* yang terdapat pada *hardware* dan digunakan untuk menerapkan suatu sistem dan hubungan antara komponen *hardware*.

8. *Component Diagram*

   Merupakan diagram yang menggambarkan *software* pada suatu sistem.

9. *Object Diagram*

   Merupakan diagram yang menggambarkan obyek – obyek pada suatu sistem dan hubungan antaranya.

10. *Composite Structure Diagram*

    Merupakan diagram yang menggambarkan struktur internal dari pengklasifikasian dan termasuk titik–titik interaksi penklasifikasian kebagian lainnya dari suatu sistem.

11. *Interaction Overview Diagram*

    Merupakan diagram untuk memvisualisasikan kerjasama dan hubungan antara *activity diagram* dengan *sequence diagram.*

12. *Package Diagram*

Merupakan diagram yang digunakan untuk mengelompokan kelas dan juga menunjukan bagaimana elemen model akan disusun serta menggambarkan ketergantungan antara paket–paket.

13. *Diagram Timing*

Merupakan diagram yang disebut sebagai bentuk lain dari interaksi diagram yang fokus utamanya adalah waktu.

2.11 Protégé

Protégé merupakan sebuah perangkat lunak *open source* yang digunakan untuk pengembangan sistem yang bertindak sebagai sebuah sistem manajemen pengetahuan. Aplikasi yang dikembangkan dengan Protégé biasa digunakan dalam pemecahan masalah dan pembuatan keputusan dalam sebuah domain (Protégé Team, 2017).

Gambar 2. 3. Logo Protégé

Sumber: (Protégé Team, 2017)

Protégé ini dikembangkan oleh sebuah tim yang berada pada *Standford Center for Biomedical Informatics Research*, dan *Plug-in* dari *tools Protégé* dapat digunakan untuk membangun aplikasi berbasis *ontology* baik yang sederhana maupun yang kompleks.