PERANCANGAN SISTEM PAKAR 6

6.1 PENDEFINISIAN MASALAH

Sebelum mulai membangun sistem pakar, permasalahan harus didefinisikan dengan tepat. Layaknya proyek pengembangan perangkat lunak lainnya, ada beberapa hal yang harus dipertimbangkan. Pertimbangan ini merupakan suatu yang tipikal dari manajemen proyek pada program konvensional, namun pada pengembangan sistem pakar perlu disesuaikan dengan kebutuhan khusus Pada bagan berikut ini tergambar sudut pandang manajemen tingkat tinggi mengenai pengembangan sistem pakar.

Pertimbangan khusus dalam pengembangan sistem pakar berikut ini disajikan dalam bentuk pertanyaan dan jawaban sebagai acuan / petunjuk (guideline) dalam proyek-proyek sistem pakar.

6.1.1 Menentukan Paradigma Yang Tepat

Mengapa Kita Membangun Sistem Pakar ?

Pertanyaan ini mungkin pertanyaan paling penting yang perlu ditanyakan dalam setiap proyek sistem pakar. Pertanyaan ini semestinya dijawab oleh pemilik (*owners*) atau pemegang saham (*stockholders*) yang membiayai pengembangan sistem pakar. Sebelum mulai, harus ada identifikasi yang jelas mengenai apa masalah yang dihadapi (*problem*), siapa pakar (*experts*), dan siapa yang akan menggunakan (*users*)

6.1.2 Hasil Yang Diharapkan (*Payoff*)

Apa Hasil Yang Diharapkan ?

Pertanyaan ini terkait dengan pertanyaan pertama. Pertanyaan ini lebih bersifat pragmatis yang terkait dengan pengembalian investasi (*return on investment*) sumber daya manusia, waktu, uang dan sumber daya lainnya. Hasil yang diperoleh mungkin berupa uang, peningkatan efisiensi, atau keuntungan lain dari sistem pakar. Namun jika tidak ada seorangpun yang menggunakan sistem ini maka dikatakan tidak ada hasil yang diperoleh.

6.1.3 Peralatan (*Tools*)

Peralatan Apa Yang Tersedia Untuk Membangun Sistem Pakar ?

Saat ini terdapat beberapa peralatan sistem pakar dengan kelebihan dan kekurangan masing-masing. Peralatan ini juga berkembang dengan pesat. Salah satu cara untuk memilih peralatan mana

yang digunakan nantinya adalah dengan membaca / menelusuri literatur terkini dan bertanya pada orang-orang yang pernah mebangun sistem pakar.

6.1.4 Biaya (*Cost*)

Berapa Besarnya Biaya Yang Dibutuhkan ?

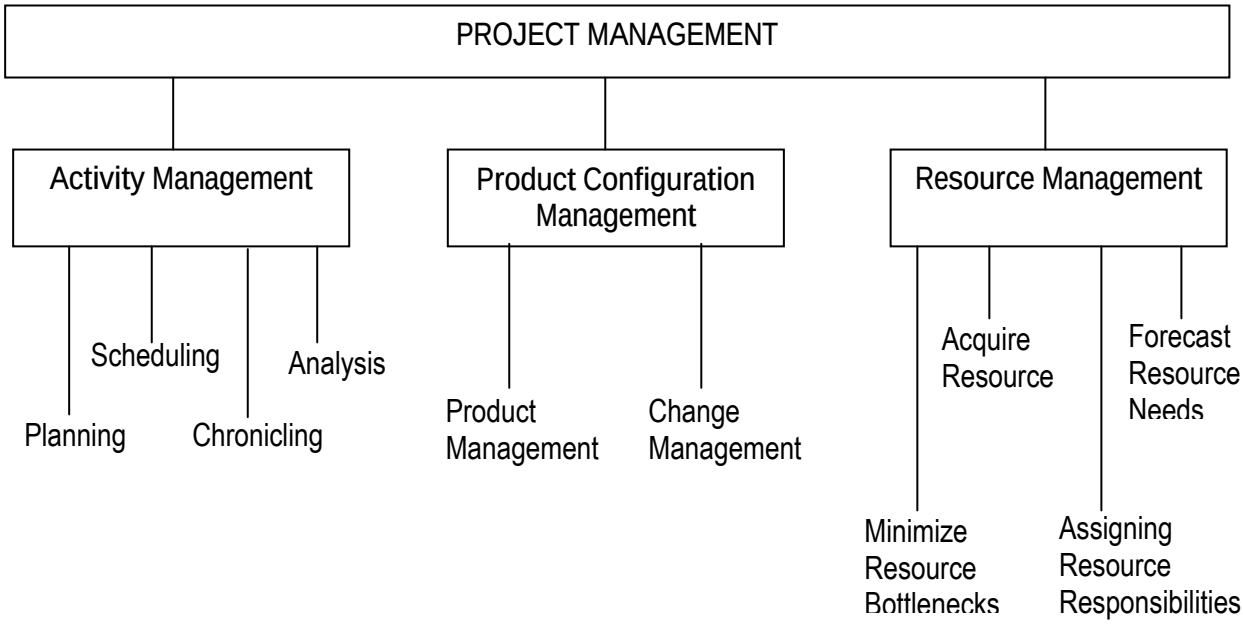
Besarnya biaya yang dibutuhkan untuk membangun sistem pakar bergantung pada sumber daya manusia, waktu, dan sumber daya lainnya yang digunakan. Selain hardware dan software yang dibutuhkan untuk menjalankan peralatan sistem pakar, perlu juga dipertimbangkan biaya untuk pelatihan, terutama jika personel yang terlibat tidak atau memiliki sedikit pengalaman dalam menggunakan peralatan sistem pakar.

6.2. TAHAPAN PENGEMBANGAN SISTEM PAKAR

Pengembangan sebuah sistem pakar akan bergantung pada sumber daya yang tersedia. Seperti halnya proyek-proyek pengembangan perangkat lunak lainnya, pengembangan sistem juga akan bergantung pada pengelolaan proses pengembangan.

6.2.1 Manajemen Proyek (*Project Management*)

Manajemen proyek ditujukan untuk melaksanakan kegiatan-kegiatan berikut, seperti yang tergambar pada gambar 6.1 di bawah ini :



Gambar 6.1. Tugas Manajemen Proyek

Manajemen Aktifitas (*Activity Management*)

- Perencanaan (*Planning*)
  - Mendefinisikan aktifitas.
  - Mendefinisikan prioritas aktifitas.
  - Mendefinisikan kebutuhan sumber daya.
  - Mendefinisikan jangka waktu pelaksanaan.
  - Mendefinisikan tanggung jawab.
- Penjadwalan (*Scheduling*)
  - Menentukan awal dan akhir pelaksanaan.
  - Menyelesaikan perselisihan penjadwalan tugas yang memiliki prioritas sama.
- Monitoring (*Chronicling*)
  - Memonitor kinerja proyek.
- Analisis (*Analysis*)
  - Menganalisis rencana, jadwal, dan kegiatan monitoring.

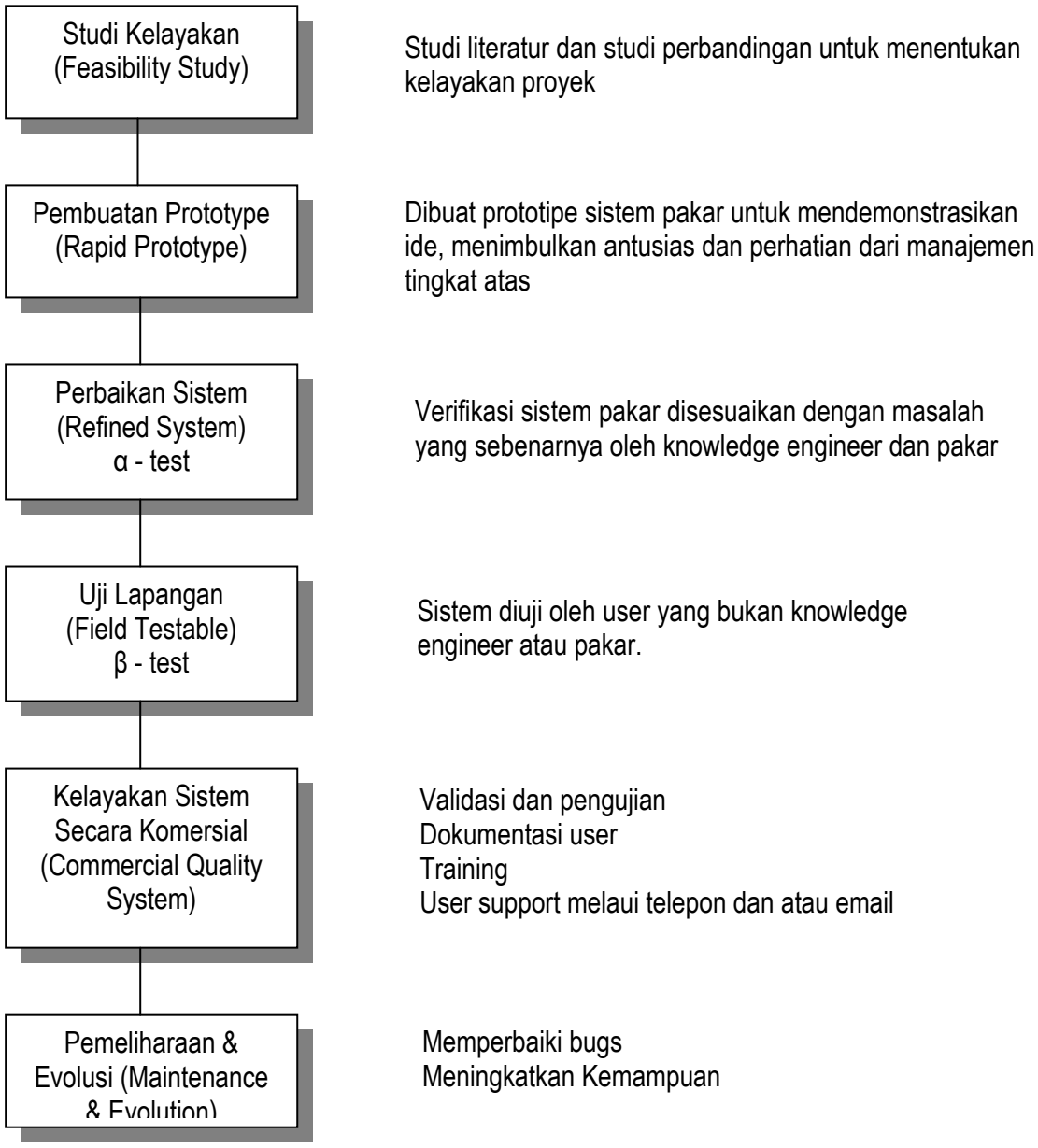
Manajemen Konfigurasi Produk (*Product Configuration Management*)

- Manajemen produk (*Product Management*)
  - Mengelola versi produk yang dihasilkan.
- Manajemen perubahan (*Change Management*)
  - Mengelola proposal perubahan dan evaluasi dampak.
  - Menugaskan personel untuak melakukan perubahan.
  - Menginstal versi produk terbaru.

Manajemen Sumber Daya (Resource Management)

- Meramalkan kebutuhan sumber daya
- Mendapatkan sumber daya
- Mengatur tanggung jawab untuk penggunaan sumber daya secara optimum
- Menyediakan sumber daya kritis untuk meminimisasi hambatan

Pada bagan berikut ini digambarkan tahapan yang dilalui dalam pengembangan sistem pakar secara umum.



Gambar 6.2 Tahapan Pengembangan Sistem Pakar Secara Umum

6.2.2 Masalah Implementasi (*Delivery Problem*)

Setelah pengembangan, seringkali implementasi sistem pakar memerlukan biaya hardware dan software yang besar. Ditambah lagi dengan biaya pemeliharaan setiap tahunnya. Bergantung pada jumlah sistem pakar yang diimplementasikan, masalah ini dapat menjadi masalah yang besar dalam proses pengembangan, dan harus dipertimbangkan pada saat awal. Idealnya sistem pakar harus dapat

berjalan pada hardware standar sehingga tidak perlu hardware khusus yang tentunya akan meningkatkan biaya.

Pada beberapa kasus, sistem pakar harus diintegrasikan dengan program yang telah ada sebelumnya. Dalam hal ini perlu dipertimbangakan komunikasi dan koordinasi antara input/output dari sistem pakar dengan program-program tersebut.

6.2.3 Pemeliharaan dan Evolusi (*Maintenance & Evolution*)

Pemeliharaan dan evolusi sistem pakar merupakan aktifitas yang lebih bersifat terbuka (tidak pernah berhenti – *open-ended*) dibandingkan dengan program konvensional, yang . Karena sistem pakar tidak didasarkan pada algoritma, kinerjanya ditentukan oleh knowledge yang terdapat di dalamnya. Kinerja sistem dapat meningkat saat knowledge baru diperoleh dan knowledge yang lama dimodifikasi. Pada produk komersial, harus ada mekanisme yang sistematis dan efisien untuk mengumpulkan laporan kesalahan (*bugs report*). Proses pemeliharaan dapat berjalan baik jika terdapat *bugs report*.

Peningkatan kemampuan sistem pakar setelah diimplementasikan juga merupakan perhatian khusus dalam produk komersial. Pengembang harus memperhatikan kebutuhan dan keinginan user dan melakukan perbaikan.

6.3. KESALAHAN PADA TAHAPAN PENGEMBANGAN

Beberapa kesalahan yang mungkin terjadi pada pengembangan sistem pakar adalah :

- Kesalahan knowledge pakar (*Expert's knowledge error*). Pakar merupakan sumber knowledge. Jika knowledge yang berasal dari pakar mengandung kesalahan maka kesalahan berlanjut pada keseluruhan proses pengembangan sistem. Untuk proyek yang melibatkan resiko hidup manusia dan properti, perlu dibuat suatu prosedur formal untuk menilai knowledge dari pakar. Misalkan dengan membentuk sebuah panel yang beranggotakan user, pakar independen dari bidang yang sama, pengembang sistem, dan manajer yang akan melakukan review serta analisis terhadap pemecahan solusi dan tehnik yang digunakan untuk membangun sistem.

  Keuntungan dari panel ini adalah knowledge dari pakar akan dievaluasi validitas dan akurasinya sejak awal pengembangan. Semakin banyak kesalahan yang ditemukan akan semakin besar biaya yang dibutuhkan untuk memperbaiki. Jika knowledge tidak diverifikasi di tahap awal, maka pengujian utama dilakukan pada tahap validasi final sistem yang memeriksa

apakah sistem memenuhi semua kebutuhan terutama dari segi ketepatan dan kelengkapan solusi. Kerugian dari pembentukan panel ini adalah dalam hal munculnya biaya tambahan.

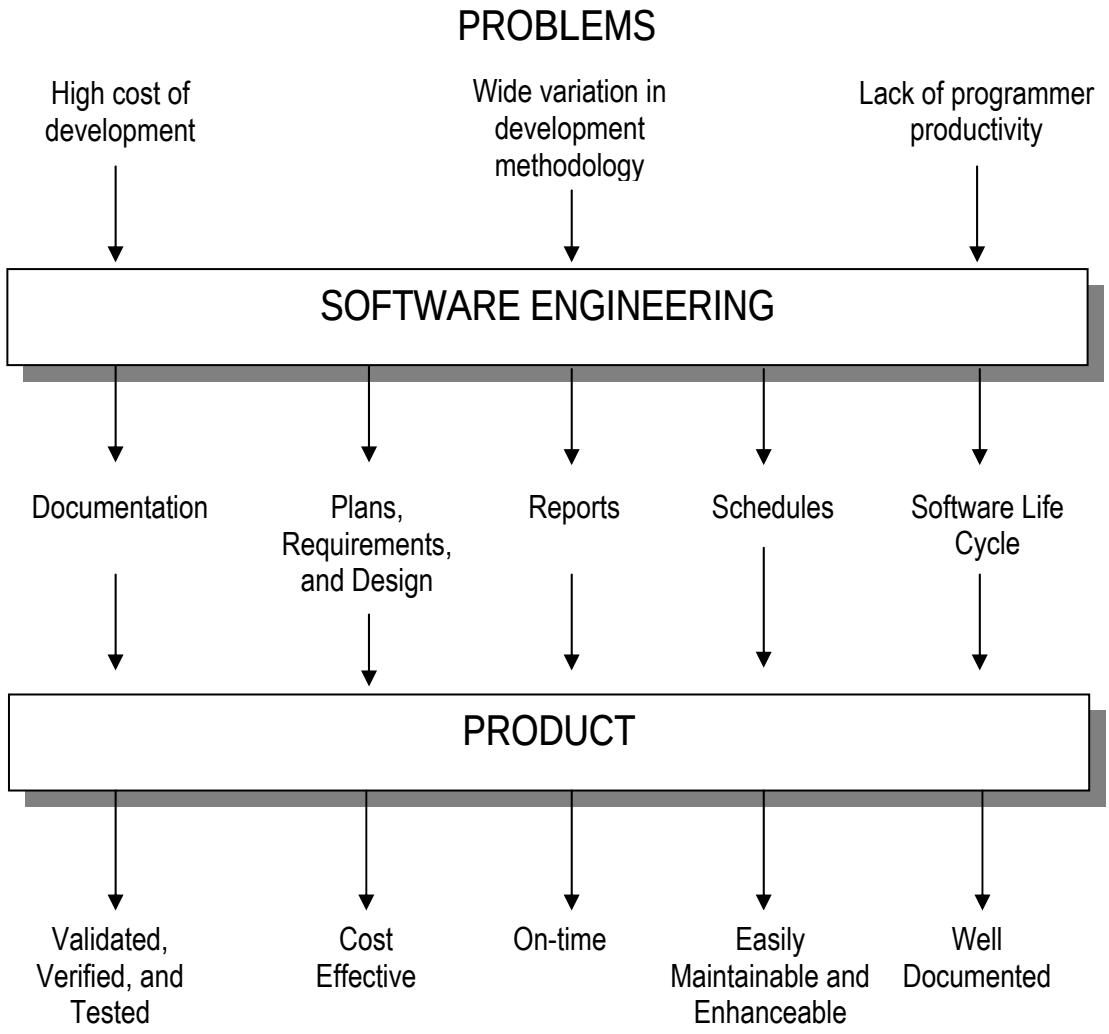
- Kesalahan semantik (*Semantic error*). Kesalahan semantik terjadi jika arti dari knowledge tidak dikomunikasikan secara tepat. Hal ini dapat disebabkan karena knowledge engineer salah mengintepretasikan jawaban / knowledge yang diberikan oleh pakar atau si pakar salah menangkap pertanyaan yang diberikan oleh knowledge engineer ataupun keduanya. Misalkan, si pakar memberikan knowledge : "You can extinguish a fire with water" dan knowledge engineer menganggapnya sebagai " All fires can be extinguished by water".
- Kesalahan sintaks (*Syntax error*). Kesalahan ini terjadi jika bentuk aturan atau fakta yang tidak tepat dimasukkan ke dalam sistem. Tool sistem pakar harus memberi tanda (*flag*) pada kesalahan ini dan memberikan pesan yang sesuai. Kesalahan lain yang terjadi pada tahap pengembangan knowledge-base disebabkan pada kesalahan sumber knowledge yang tidak dideteksi pada tahap awal.
- Kesalahan mesin inferensi (*Inference engine error*). Sama seperti software lainnya, mesin inferensi juga dapat mengandung bug/error. Pada saat sistem pakar akan diimplementasikan, semua bugs harus sudah diperbaiki. Namun, kadang terdapat bugs yang muncul pada kondisi yang khusus/langka. Secara umum, bugs mesin inferensi dapat muncul pada saat operasi pencocokan pola (*pattern matching*), konflik, dan eksekusi, dan akan sulit dideteksi jika bugs ini tidak konsisten.

  Metode sederhana yang dapat digunakan untuk memeriksa error adalah dengan bertanya pada user lain atau vendor. Vendor harus menyediakan daftar pelanggan, bugs report, dan cara perbaikan bugs. Kelompok user juga merupakan sumber informasi yang baik untuk menangani bugs ini.
- Kesalahan rantai proses inferensi (i*nference chain error*). Kesalahan ini dapat disebabkan oleh kesalahan knowledge, kesalahan semantik, bugs inferensi engine, spesifikasi prioritas aturan (*rule*) yang tidak tepat, dan interaksi antar rule yang tidak diperhitungkan. Kesalahan rantai proses inferensi yang lebih kompleks disebabkan karena ketidakpastian rule dan fakta, dan akibat lanjutan dari ketidakpastian tersebut dalam rantai proses inferensi dan nonmonoticity.
- Batas toleransi terhadap kekurangan (*Limits of ignorance error*). Salah satu masalah yang umum dihadapi semua tahap pengembangan adalah menentukan batas toleransi terhadap kekurangan oleh sistem. Pakar mengetahui batas pengetahuan yang mereka miliki dan kemampuannya berkurang pada batas tersebut. Pakar harus jujur mengakui bahwa solusi

mereka tidak optimal dan mengandung ketidakpastian. Pada sistem pakar, kecuali sistem dirancang khusus untuk mengakui ketidakpastian, sistem akan tetap memberikan solusi walaupun proses inferensi yang dilakukan dan fakta yang dimiliki sangat sedikit dan lemah.

6.4. REKAYASA PERANGKAT LUNAK (*SOFTWARE ENGINEERING*) DAN SISTEM PAKAR

Sistem pakar harus diberlakukan seperti halnya perangkat lunak lain seperti word processor, program payroll, computer game dan lainnya, yang mengimplementasikan metodologi rekayasa perangkat lunak (*software engineering*) untuk mengembangkan perangkat lunak berkualitas. Namun ada perbedaan yang signifikan dari misi sistem pakar dengan program lainnya. Sistem pakar memiliki misi untuk menyediakan kepakaran / knowledge dengan kemampuan bekerja yang tinggi dan memiliki kemungkinan mengandung resiko membahayakan kehidupan manusia dan properti. Oleh karena itu, sistem pakar merupakan sistem dengan kemampuan kerja yang tinggi dan harus memiliki kualitas yang baik. Rekayasa perangkat lunak (*software engineering*) memberikan metodologi yang digunakan untuk membangun software yang berkualitas seperti yang tergambar berikut ini :



Gambar 6.3 Metodologi Rekayasa Perangkat Lunak

Kualitas merupakan suatu istilah yang sulit untuk dideskripsikan secara umum karena memiliki arti yang berbeda pada orang yang berbeda. Salah satu definisi kualitas adalah atribut suatu obyek yang diinginkan atau yang dibutuhkan dan dinyatakan dalam suatu skala ukuran. Obyek yang dimaksud disini adalah software atau hardware, dan atribut serta nilainya dikatakan sebagai metrik (metric). Tabel 6.1 berikut ini berisi beberapa metric yang dapat digunakan untuk menilai kualitas sebuah sistem pakar. Metric ini hanya sebagai panduan, tidak semua metric yang terdapat pada tabel ini dapat diukur pada suatu sistem pakar, atau malah ada metric yang sebenarnya perlu diukur namun tidak terdapat pada tabel ini.

Tabel 6.1 Metric Kualitas untuk Sistem Pakar

- Output yang tepat yang dihasilkan dari input yang tepat (*Correct output given correct input*)
- Output yang lengkap yang dihailkan dari input yang tepat (*Complete output given correct*)
- Output yang konsisten dari input yang sama (*Consistent output given the same input again*)
- Dapat diandalkan dan tidak sering crash karena bugs (*Reliable so that it does not crash (often due to bugs*)
- Mudah digunakan dan user-friendly (*Usable for people and preferably user-friendly*)
- Mudah pemeliharaannya (*Maintainable*)
- Dapat ditingkatkan (*Enhanceable*)
- Telah divalidasi memenuhi kebutuhan dan permintaan user (*Validated to prove it satifies the user's needs and requests*)
- Telah diuji ketepatan dan kelengkapannya (*Tested to prove correctnes and completeness*)
- Murah (*Cost-effective*)
- Koding yang dapat digunakan kembali untuk aplikasi lain (*Reuseable code for other applications*)
- Portable to other hardware/software environments
- Dapat berinteraksi dengan software lain (*Interfaceable with other software*)
- Koding yang mudah dimengerti (*Understandable code*)
- Ketepatan (*Accurate*)
- Presisi (*Precise*)
- Knowledge base yang terverifikasi (*Verified knowledge-base*)
- Fasilitan penjelasan (*Explanation facility*)

Daftar metric yang kita buat akan mempermudah untuk menentukan metric mana yang menjadi prioritas, karena kemungkinan ada satu atau lebih metric yang saling konflik.

6.5. SIKLUS HIDUP SISTEM PAKAR (*EXPERT SYSTEM LIFE CYCLE*)

Salah satu metode utama dalam rekayasa perangkat lunak (software engineering) adalah siklus hidup (life cycle). Siklus hidup suatu software adalah satu periode waktu mulai dari pembentukan konsep awal dan berakhir hingga software tersebut tidak digunakan lagi. Konsep siklus hidup ini tidak memisahkan tahapan pengembangan dengan pemeliharaan, melainkan merupakan satu kesatuan yang menghubungkan semua tahap. Merencanakan pemeliharaan dan evolusi pada tahap awal siklus hidup akan mengurangi biaya proses pemeliharaan dan evolusi nantinya.

6.5.1 Biaya Pemeliharaan (*Maintenance Cost*)

Pada program konvensional, besarnya biaya pemeliharaan biasanya berkisar antara 60 – 80 % dari total biaya software dan dua sampai empat kali biaya pengembangan. Meskipun belum ada informasi mengenai berapa besarnya biaya pemeliharaan software sistem pakar, namun diperkirakan jumlahnya akan lebih besar dari program konvensional. Jika program konvensional yang memiliki algoritnma yang telah diketahui membutuhkan biaya pemeliharaan yang demikian besarnya, maka sistem pakar yang dibuat berdasarkan knowledge yang kebanyakan bersifat heuristik diperkirakan akan membutuhkan biaya lebih besar. Terutama jika sistem pakar melakukan inferensi lebih banyak dalam kondisi ketidakpastian maka biaya pemeliharaan dan evolusinya akan bertambah besar.

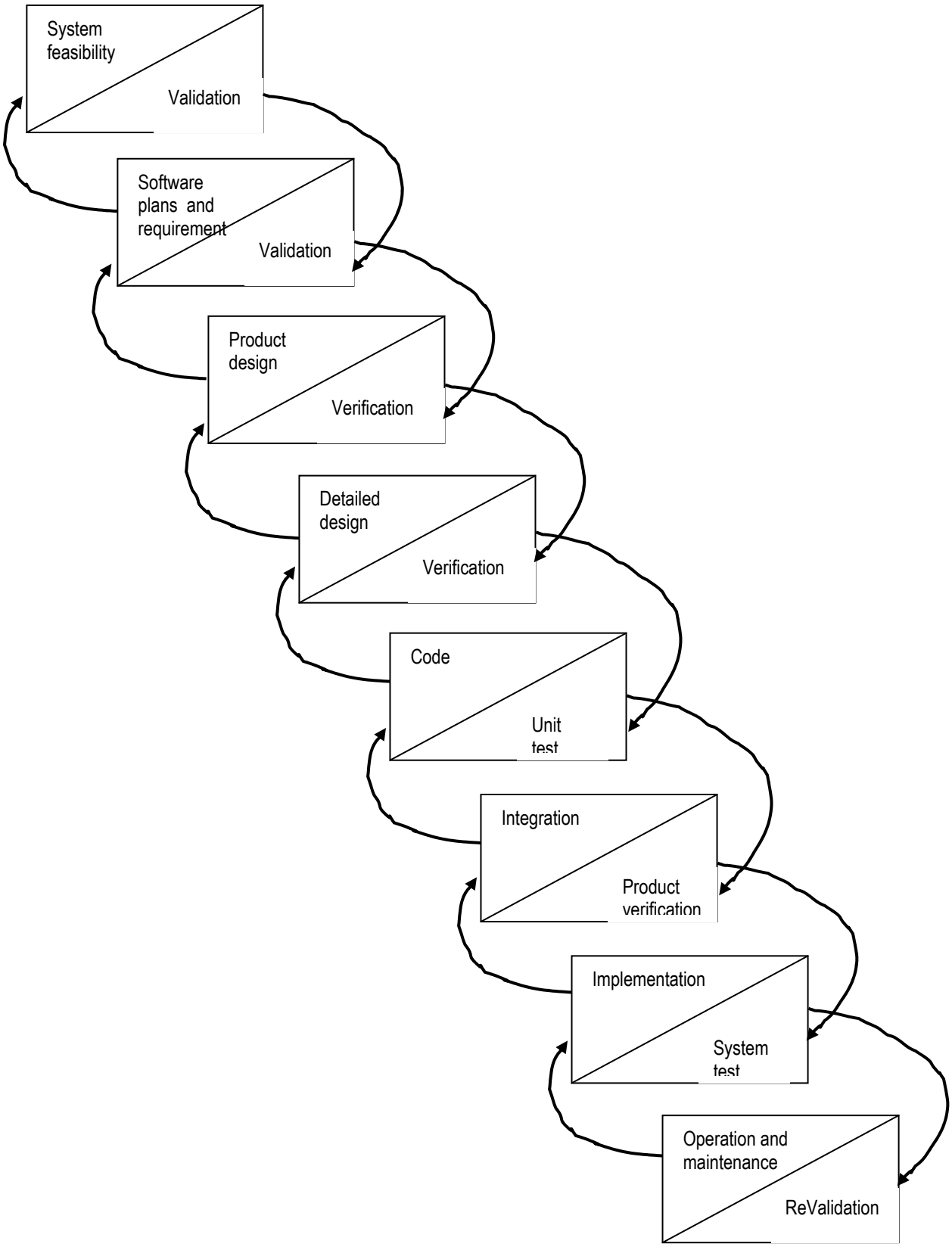
6.5.2 Model Air Tejun (*Waterfall Model*)

Sejumlah model siklus hidup software telah dikembangkan bagi program konvensional. Salah satu model yang klasik yang paling dikenal oleh para programer adalah model air terjun (*waterfall model*) seperti yang digambarkan pada gambar 6.4. Pada model ini, setiap tahapnya diakhiri dengan validasi dan verifikasi untuk meminimalkan masalah yang mungkin terjadi pada tiap tahapannya. Pada gambar juga terlihat panah yang menuju ke tahap berikutnya serta panah yang memungkinkan kembali dari satu tahap ke tahap sebelumnya. Hal ini menggambarkan pengembangan secara iteratif antara dua tahapan siklus hidup. Bentuk iterasi ini akan meminimalkan biaya dibandingkan dengan jika iterasi dapat dilakukan untuk beberapa tahap siklus.

Istilah lain yang juga dipergunakan untuk siklus hidup adalah model proses (*process model*), karena model ini memperhatikan dua isu dasar dari pengembangan software, yaitu :

(1) Apa yang harus dilakukan berikutnya ?

(2) Berapa lama tahap berikutnya akan dilaksanakan ?



Gambar 6.4 Model Waterfall

6.5.3 Model Code-and-Fix

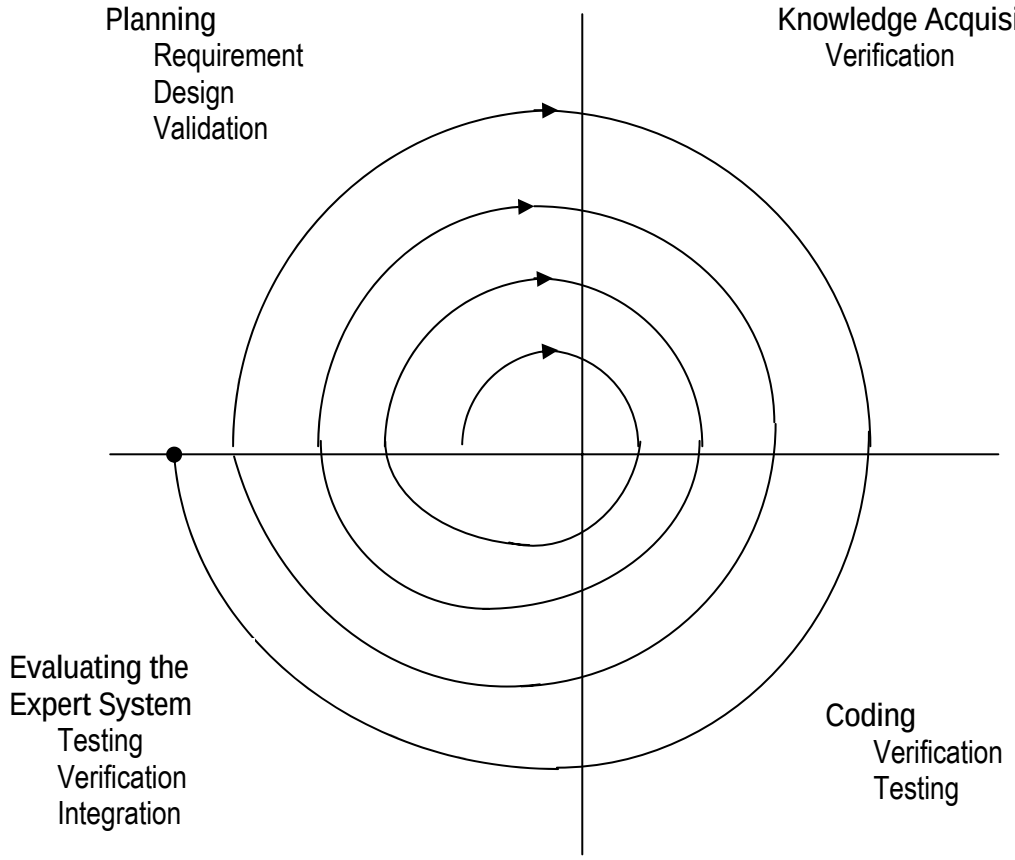
Model ini mengembangkan software dengan cara membuat program dan kemudian diperbaiki jika terdapat kesalahan. Model ini merupakan model awal yang digunakan untuk mengembangkan software. Namun sejak tahun 1970-an, model ini mulai ditinggalkan dan dikembangkan model waterfall yang memberikan metodologi lebih sistematik dan sangat membantu terutama pada proyek-proyek yang besar. Namun kesulitan pada model waterfall adalah perlu adanya informasi yang lengkap pada setiap tahapnya, dan bukan sesuatu hal yang mudah untuk mendapatkan informasi tersebut. Pada prakteknya, sering tidak mungkin untuk menulis dokumentasi kebutuhan yang lengkap sebelum dibangun prototipe. Sehingga yang terjadi adalah "kerja dua kali", membuat prototipe, kemudian dari prototipe diperoleh informasi kebutuhan dan barulah dibangun sistem final.

6.5.4 Model Incremental

Model incremental (*Incremental waterfall model*) merupakan perbaikan dari model waterfall dan sebagai standar pendekatan top-down. Ide dasar dari model ini adalah membangun software secara meningkat (increment) berdasarkan kemampuan fungsional. Model incremental ini diaplikasikan pada sistem pakar dengan penambahan rules yang mengakibatkan bertambahnya kemampuan fungsional sistem. Keuntungan dari model ini adalah bahwa penambahan kemampuan fungsional akan lebih mudah diuji, diverifikasi, dan divalidasi dan dapat menurunkan biaya yang dikeluarkan untuk memperbaiki sistem. Model incremental merupakan model continous rapid prototype dengan durasi yang diperpanjang hingga akhir proses pengembangan. Pada model prototipe biasa, prototipe hanya dibuat pada tahap awal untuk mendapatkan kebutuhan user.

6.5.5 Model Spiral

Salah satu cara untuk memvisualisasikan model incremental adalah dengan mengadaptasi model spiral konvensional seperti pada gambar 6.5. Setiap lintasan pada gambar spiral menambahkan kemampuan fungsional pada sistem. Poin akhir yang diberi label "delivered System" sesungguhnya bukan merupakan akhir dari lintasan spiral, melainkan merupakan awal spiral baru yang dimulai dengan pemeliharaan dan evolusi (*maintenance and evolution*) dari sistem.



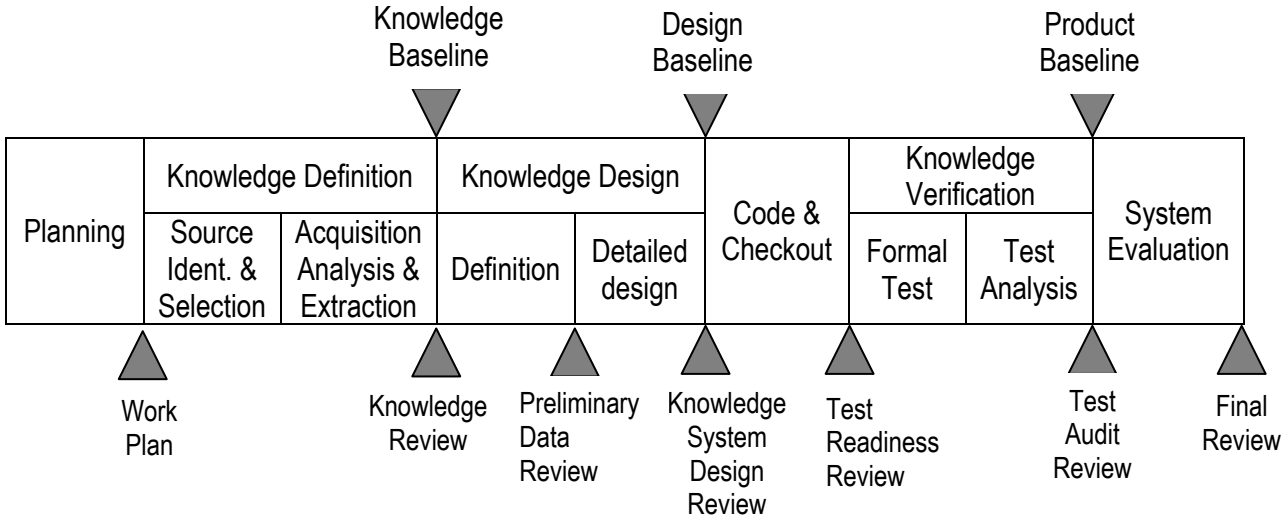
Gambar 6. 5 Model Pengembangan Sistem Pakar Berbentuk Spiral

6.6 MODEL SIKLUS HIDUP LINIER

Model siklus hidup yang telah berhasil diterapkan pada sejumlah proyek pengembangan sistem pakar adalah model linier, seperti yang diilustrasikan pada gambar 6.6. Siklus ini terdiri dari sejumlah tahap mulai dari perencanaan (*planning*) hingga evaluasi sistem (*system evaluation*) dan akan berulang hingga sistem diimplementasikan, yang kemudian sistem akan memasuki tahap pemeliharaan dan evolusi. Walaupun tidak digambarkan secara eksplisit, proses verifikasi dan validasi dijalankan secara paralel di setiap tahap. Masing-masing tahapan terdiri dari beberapa tugas (*task*). Tidak semua task suatu tahap perlu dilaksanakan, bergantung pada tipe aplikasi yang dibangun.

6.6.1 Perencanaan (*Planning*)

Tujuan dari tahap perencanaan adalah untuk menghasilkan rencana kerja (*work plan*) formal untuk pengembangan sistm pakar. Rencana kerja adalah sekumpulan dokumen yang digunakan untuk menuntun dan mengevaluasi proses pengembangan. Tugas (*task*) yang dilakukan pada tahap ini dapat dilihat pada tabel 6.2.



Gambar 6.6 Model Linier Siklus Hidup Pengembangan Sistem Pakar

Tabel 6.2 Tugas Tahap Perencanaan

Tugas	Tujuan
Penilaian kelayakan (*feasibility assessment*)	Menentukan apakah perlu dibangun suatu sistem, jika ya, apakah sistem tersebut mengaplikasikan sistem pakar
Pengelolaan sumber daya (*resource management*)	Menilai sumber daya manusia, waktu, uang, software, hardware yang dibutuhkan. Bagaimana mendapatkan dan mengelola sumber daya tersebut.
Pembagian tugas (*task phasing*)	Menentukan tugas dan urutannya pada masing-masing tahap.
Penjadwalan (*schedules*)	Menentukan tanggal awal dan akhir pelaksanaan tugas-tugas dalam masing-masing tahap.
Penggambaran fungsi awal sistem (*preliminary functional layout*)	Mendefinisikan sistem yang akan dibuat dengan menentukan fungsi sistem dan menentukan tujuan sistem.
Kebutuhan level atas (*high-level requirements*)	Mendeskripsikan pandangan level atas mengenai bagaimana fungsi sistem akan dicapai.

6.6.2 Pendefinisian Knowledge (*Knowledge Definition*)

Tujuan tahap ini adalah mendefiniskan kebutuhan knowledge dari sistem pakar. Tahap ini terdiri dari dua tugas utama yaitu (1) Identifikasi dan pemilihan sumber knowledge (*Knowledge source*

identification and selection) dan (2) Perolehan, analisis dan ekstraksi knowledge (*Knowledge acquisition, analysis, and extraction*). Masing-masing tugas utama ini terdiri dari beberapa tugas seperti pada tabel 6.3 dan 6.4 berikut.

Tabel 6.3 Tugas Identifikasi dan Pemilihan Sumber Knowledge

Tugas	Tujuan
Identifikasi sumber (s*ource identification*)	Menentukan siapa dan apa sumber knowledge tanpa melihat ketersediaannya
Prioritas sumber (*source importance*)	Menentukan sumber knowledge dalam urutan prioritas
Ketersediaan sumber (*source availability*)	Membuat daftar sumber knowledge diurutkan berdasarkan ketersediaannya.
Pemilihan sumber (*source selection*)	Memilih sumber knowledge berdasarkan prioritas dan ketersediaannya

Tabel 6.4 Tugas Perolehan, Analisis dan Ekstraksi Knowledge

Tugas	Tujuan
Strategi perolehan (*acquisition strategy*)	Menentukan bagaimana knowledge akan diperoleh
Identifikasi elemen knowledge (*knowledge element identification*)	Memperoleh knowledge spesifik dari sumber yang akan digunakan
Sistem klasifikasi knowledge (*knowledge classification system*)	Mengklasifikasikan dan mengorganisasikan knowledge yang akan membantu verifikasi dan pemahaman bagi pengembang.
Gambaran fungsi detail (*Detailed functional layout*)	Menspesifikasikan kemampuan fungsional sistem secara detail. Level ini lebih bersifat tehnikal dibandingkan dengan preliminary functional layout yang bersifat manajerial
Alur kendali awal (*preliminary control flow*)	Mendeskripsikan fase umum yang akan dilalui sistem pakar. Fase yang dimaksud adalah kelompok logika dari rule yang diaktifkan atau dinon-aktifkan untuk mengatur alur eksekusi
Panduan user awal (*preliminary user's manual*)	Mendeskripsikan sistem dari sudut pandang user. Tugas ini penting dilakukan untuk melibatkan user dan mendapatkan feedback.
Spesifikasi kebutuhan (*requirements specification*)	Mendefinisikan apa yang akan dilakukan oleh sistem. Nantinya sistem pakar akan divalidasi menggunakan

	spesifikasi ini.
Baseline knowledge	Basis knowledge untuk sistem pakar. Setelah tahap ini, segala perubahan harus dilakukan secara formal. Formulasi knowledge yang dihasilkan tahap ini harus sudah tepat dan sesuai dengan tahapan berikutnya.

6.6.3 Perancangan Knowledge (*Knowledge Design*)

Tujuan tahap ini adalah menghasilkan rancangan rinci untuk sistem pakar. Tahap ini terdiri dari dua tahap utama, yaitu (1) Pendefinisian knowledge (*knowledge definition*) dan (2) Perancangan rinci (*detailed design*). Masing-masing tugas utama ini terdiri dari beberapa tugas seperti pada tabel 6.5 dan 6.6 berikut.

Tabel 6.5 Tugas Pendefinisian knowledge

Tugas	Tujuan
Representasi knowledge (*knowledge representation*)	Menspesifikasikan bagaimana knowledge disajikan seperti rule, frame, atau logika, bergantung dari tool apa yang akan digunakan
Struktur kendali rinci (d*etailed control structure*)	Menspesifikasikan tiga struktur kendali, (1) Jika sistem digabungkan dengan prosedur lain, bagaimana sistem pakar akan dipanggil, (2) Kontrol dari kelompok rule yang berhubungan dalam suatu sistem, (3) struktur kendali metalevel.
Struktur fakta internal (*internal fact structure*)	Menspesifikasikan struktur fakta internal secara konsisten untuk membantu pemahaman.
User interface awal (p*reliminary user interface*)	Menspesifikasikan user interface awal dan mendapatkan feedback dari user mengenai user interface ini.
Rencana pengujian awal (*initial test plan*)	Menspesifikasikan bagaimana program akan diuji, termasuk mendefinisikan data tes, mekanisme tes, dan bagaimana hasil tes dianalisis.

Tabel 6.6 Tugas Perancangan rinci

Tugas	Tujuan
Struktur rancangan (*design structure*)	Menspesifikasikan bagaimana knowledge diorganisasikan dalan knowledge base dan apa saja

	yang terkandung pada knowledge base
Strategi implementasi (*implementation strategy*)	Menspesifikasikan bagaimana sistem akan diimplementasikan
Interface user rinci (*detailed user interface*)	Menspesifikasikan user interface yang rinci setelah menerima feedback dari user
Laporan dan spesifikasi rancangan (*design spesifications and report*)	Mendokumentasikan perancangan.
Rencana pengujian rinci (*detailed test plan*)	Menspesifikasikan bagaimana program akan diuji dan diverifikasi.

6.6.4 Koding dan pengujian (*Code and Checkout*)

Tahap ini menandakan dimulainya pemrograman. Tahap ini diakhiri dengan adanya dokumen test readiness review yang menentukan apakah sistem pakar siap untuk tahap selanjutnya dari verifikasi knowledge. Tahap ini terdiri dari beberapa tugas seperti yang tercantum pada tabel 6.7 berikut.

Tabel 6.7 Tugas Koding dan Pengujian

Tugas	Tujuan
Koding (*coding*)	Membuat program
Pengujian (*test*)	Menguji koding menggunakan data tes, mekanisme tes dan prosedur analisis hasil tes
Dokumen program (*source listing*)	Menghasilkan dokumentasi source code
User manual	Menghasilkan user manual sehingga user dan pakar dapat memberikan feedback
Installation / operations guide	Mendokumentasikan instalasi / operasi sistem untuk user
Dokumen deskripsi sistem (*system description document*)	Mendokumentasikan sistem pakar secara keseluruhan meliputi fungsionalitas, batasan, dan masalah

6.6.5 Verifikasi Knowledge (*Knowledge Verification*)

Tahap ini bertujuan untuk menentukan ketepatan, kelengkapan, dan konsistensi sistem. Tahap ini terdiri dari dua tugas utama yaitu (1) tes formal, dan (2) analisis tes, yang masing-masing terdiri dari beberapa tugas seperti yang tercantum pada tabel 6.8 dan 6.9 berikut.

Tabel 6.8 Tugas Tes Formal

Tugas	Tujuan
Prosedur tes (*test procedure*)	Mengimplementasikan prosedur formal tes
Laporan pengujian (*tes reports*)	Mendokumentasikan hasil tes

Tabel 6.9 Tugas Analisis Tes

Tugas	Tujuan
Evaluasi hasil (*result evaluation*)	Menganalisis hasil tes
Rekomendasi (*recommendation*)	Mendokumentasikan rekomendasi dan kesimpulan hasil tes

6.6.6 Evaluasi sistem (*System Evaluation*)

Tahap ini merupakan tahap akhir dari siklus dan bertujuan untuk menyimpulkan apa yang dipelajari dari rekomendasi untuk perbaikan dan peningkatan. Tahap ini beberapa tugas seperti yang tercantum pada tabel 6.10 berikut.

Tabel 6.10 Tugas Evaluasi Sistem

Tugas	Tujuan
Evaluasi hasil (*result evaluation*)	Mneyimpulkan hasil tes dan verifikasi
Rekomendasi (*recommendation*)	Merekomendasikan perubahan terhadap sistem
Validasi (*validation*)	Memvalidasi bahwa sistem benar sesuai dengan kebutuhan dan permintaan user
Laporan akhir (*final report*)	Menghasilkan laporan akhir.