# PANDUAN PENGGUNAAN DAN PENGISIAN SPESIFIKASI KEBUTUHAN PERANGKAT LUNAK (SKPL)

# Daftar Isi

# 1. Pendahuluan

Dokumen ini akan berisi penjelasan pemakaian dan penulisan dokumen Spesifikasi Kebutuhan Perangkat Lunak (SKPL) atau *Software Requirement Specification (SRS)*. Untuk penamaan dokumen ini selanjutnya akan digunakan istilah SKPL. Pada dasarnya SKPL adalah suatu dokumen yang menyatakan kebutuhan perangkat lunak sebagai hasil dari proses analisis yang dilakukan dalam konteks pengembangan perangkat lunak.

Dokumen ini digunakan untuk acuan dalam menulis SKPL. Akan diberikan juga beberapa *outline* dari SKPL. Detil penjelasan *outline* SKPL untuk kedua orientasi pengembangan perangkat lunak (berorientasi proses dan berorientasi objek) dijelaskan pada dokumen terpisah (Panduan GL01A dan Panduan GL01B). Dokumen ini dibuat untuk membantu membuat spesifikasi perangkat lunak yang akan dikembangkan. Isi dari dokumen ini sebagian besar adalah terjemahan dari dokumen IEEE Std 830-1993.

# 2. Definisi dan Singkatan (Akronim)

| | |
|---|---|
| SKPL | Spesifikasi Kebutuhan Perangkat Lunak |
| SRS | *Software Requirement Specification* |
| IEEE | The Institute of Electrical and Electronics Engineers |
| ANSI | American National Standard Institution |
| Std | *Standard* |

Definisi dari istilah yang akan digunakan pada dokumen ini dibuat berdasarkan hasil terjemahan dari IEEE Std 610.12-1990.

1. Kontrak
   Adalah suatu dokumen legal yang disetujui oleh pelanggan dan pengembang. Dalam kontrak akan disertai dengan kebutuhan teknis, organisasi, biaya dan jadwal pengerjaan produk. Kontrak dapat berisi informasi yang berguna, misalnya komitmen dan harapan dari organisasi yang terlibat.
2. Pelanggan
   Adalah orang atau organisasi yang membayar produk, dan biasanya (tidak harus) ia yang akan memutuskan kebutuhannya.
3. Pengembang
   Adalah orang yang menghasilkan produk untuk pelanggan.
4. Pengguna
   Adalah orang yang akan langsung menjalankan atau menggunakan produk. Pengguna dan pelanggan umumnya adalah orang yang sama, walaupun bisa juga berbeda.
5. *Baseline*
   Spesifikasi atau produk yang telah dikaji ulang (*review*) dan disetujui secara formal, yang kemudian dijadikan sebagai dasar atau basis dari pengembangan (perangkat lunak) lebih lanjut, yang hanya dapat diubah melalui suatu prosedur pengendalian perubahan formal.

## 3. Referensi


- IEEE Std. 830-1993, IEEE *Recommended Practice for Software Requirement Specifications*.
- IEEE Std. 610.12-1990, IEEE *Standard Glossary of Software Engineering Terminology* (ANSI).
- Jurusan Teknik Informatika – Institut Teknologi Bandung Panduan GL01A, Panduan Penggunaan dan Pengisian Spesifikasi Kebutuhan Perangkat Lunak Berorientasi Proses.
- Jurusan Teknik Informatika – Institut Teknologi Bandung Panduan GL01B, Panduan Penggunaan dan Pengisian Spesifikasi Kebutuhan Perangkat Lunak Berorientasi Objek.


## 4. Pertimbangan Pembuatan SKPL yang Baik

Beberapa hal yang harus diperhatikan dalam pembuatan SKPL adalah:

1. Sifat dari SKPL
2. Lingkungan SKPL
3. Karakteristik SKPL yang baik
4. Persiapan SKPL
5. Evolusi SKPL
6. Prototipe SKPL
7. Pemasukan rancangan pada SKPL
8. Pemasukan kebutuhan proyek pada SKPL

### *4.1 Sifat dari SKPL*

SKPL adalah spesifikasi dari suatu produk/program yang melakukan suatu fungsi tertentu pada lingkungan tertentu. SKPL dapat dibuat oleh wakil dari pengembang atau wakil dari pelanggan. Sebaiknya SKPL dibuat bersama-sama oleh pengembang dan pelanggan.

Penulis SKPL harus memperhatikan hal-hal berikut:

1. Fungsionalitas
   Untuk apa suatu perangkat lunak dibuat.
2. Antar muka eksternal (*External Interface*)
   Dengan apa perangkat lunak berinteraksi dengan pengguna, perangkat keras sistem, perangkat keras di luar sistem dan perangkat lunak lain.
3. Performansi
   Sejauh apa kecepatan, ketersediaan (*availability*), waktu tanggap (*response time*), waktu recovery dari berbagai fungsi perangkat lunak yang dibuat.
4. Atribut
   Seberapa tingkat portabilitas, tingkat kebenaran (*correctness*), tingkat pemeliharaan (*maintainability*), dan tingkat keamanan yang ingin dicapai.
5. Batasan perancangan
   Apakah diperlukan suatu standar, bahasa yang khusus, kebijaksanaan integritas basisdata, batasan sumberdaya, lingkungan operasi, dan lain-lain yang membatasi

pilihan-pilihan yang bisa digunakan atau keputusan-keputusan yang bisa diambil ketika perancangan.

Penulis SKPL tidak sepatutnya menuliskan spesifikasi rancangan atau kebutuhan proyek secara keseluruhan dalam SKPL. Untuk itulah penulis SKPL sepatutnya dapat membedakan hasil pekerjaan mana yang termasuk hasil analisis dan mana yang termasuk hasil perancangan.

## *4.2 Lingkungan SKPL*

Karena SKPL akan memainkan peranan penting dalam proses pengembangan perangkat lunak, penulis SKPL harus secara berhati-hati dalam memainkan peranannya (yang menuangkan hasil kerja pada suatu dokumen yang dijadikan dasar – *baseline*). Mengingat SKPL pada akhirnya akan menjadi dasar bagi kontrak antara pengembang dan pelanggan, maka suatu dokumen SKPL harus memenui syarat-syarat berikut:

1. Mendefinisikan kebutuhan perangkat lunak dengan benar. Kebutuhan perangkat lunak muncul karena ada pekerjaan yang harus diselesaikan atau karena ada karakteristik khusus dari proyek.
2. Tidak menjelaskan rancangan atau implementasi dengan rinci. Penjelasan tersebut tidak diperlukan karena bagi pengguna hal tersebut lebih teknis dan tidak perlu.
3. Tidak memaksakan penambahan suatu batasan dari perangkat lunak.

## *4.3 Karakteristik SKPL yang baik*

Karakterisitk SKPL yang baik adalah sebagai berikut:

1. Benar
2. Tidak ambigu
3. Lengkap
4. Konsisten
5. Terurut berdasarkan kepentingannya atau kestabilannya
6. Dapat diverifikasi
7. Dapat dimodifikasi
8. Dapat ditelusuri (*traceable*)

### 4.3.1 Benar

SKPL dianggap benar jika dan hanya jika setiap kebutuhan yang tercantum dalam dokumen adalah kebutuhan yang akan dipenuhi oleh perangkat lunak. Tidak ada kakas (*tools*) atau prosedur yang dapat menjamin kebenaran. Untuk memverifikasinya, SKPL harus diperbandingkan dengan spesifikasi-spesifikasi yang mendahuluinya (misalnya kontrak, *request for proposal*, *system specification*, dan lain-lain).

### 4.3.2 Tidak ambigu

SKPL tidak ambigu jika dan hanya jika setiap kebutuhan yang ditetapkan hanya memiliki satu interpretasi. Minimum kebutuhan dari setiap karakteristik produk akhir

dijelaskan dalam satu istilah yang unik.  Jika istilah tersebut memiliki banyak arti,

pada harus diberikan penjelasan. Jadi SKPL tidak boleh membingungkan baik bagi pembuat ataupun yang akan menggunakannya.

Contoh pernyataan kebutuhan yang ambigu: “Perangkat Lunak FulanSoft menampilkan data Fulan di layar monitor dengan cukup cepat”.

Contoh yang lebih baik dari pernyataan tersebut adalah: “Perangkat Lunak FulanSoft menampilkan data Fulan di layar monitor dalam waktu maksimum 3 detik setelah permintaan tampilan data diajukan (*submit*)”

#### 4.3.2.1 Kelemahan Bahasa Manusia

Kebutuhan sering ditulis dalam bahasa manusia (misalnya bahasa Indonesia atau bahasa Inggris). Bahasa manusia ini sering tidak jelas (ambigu). Bahasa yang digunakan dalam SKPL harus didiskusikan oleh pihak lain yang independen untuk mengindentifikasi ketidakjelasan dalam pemakaian bahasa.

#### 4.3.2.2 Bahasa Spesifikasi Kebutuhan

Salah satu cara untuk menghilangkan keraguan dalam penulisan bahasa ini, adalah dengan menuliskan SKPL dengan suatu bahasa spesifikasi kebutuhan yang khusus. Contohnya bahasa algoritmik, bahasa Z, *pseudo-code*. Pemroses bahasanya akan dapat mendeteksi kesalahan leksikal, sintaks atau semantik.

Kelemahan dalam menggunakan bahasa tersebut adalah waktu yang panjang untuk mempelajarinya. Juga banyak pengguna non teknis akan kesulitan. Bahasa spesifikasi kebutuhan ini cenderung akan lebih dapat digunakan pada kebutuhan khusus, dan sistem tertentu yang sangat teknis sehingga pelanggannya pun adalah pelanggan teknis.

#### 4.3.2.3 Alat Presentasi

Secara umum, kebutuhan metode dan bahasa serta alat pendukungnya, akan dibagi menjadi tiga kategori, yaitu objek, proses dan perilaku (*behaviour*).

- Pendekatan berbasis objek akan mengorganisasikan kebutuhan dalam bentuk objek-objek. Setiap objek mempunyai atribut dan layanan (fungsi) tersendiri. Objek-objek ini saling berkomunikasi melalui pemanggilan pesan. Pemanggila pesan paling sederhana dilakukan dengan meminta objek tertentu untuk menjalankan layanannya (fungsinya).
- Pendekatan berbasis proses akan mengatur kebutuhan menjadi fungsi-fungsi secara hirarkis yang saling berkomunikasi melalui suatu jalur aliran data.
- Pendekatan berbasis perilaku menjelaskan perilaku eksternal dari sistem dalam suatu pendekatan abstrak (dengan predikat kalkulus), fungsi matematika atau mesin status (*state machine*).

Manfaat pemakaian suatu kakas dan metode dalam menyiapkan SKPL bergantung pada ukuran dan kompleksitas dari program. Penggunaan berbagai pendekatan ini akan lebih baik dibarengi dengan deskripsi bahasa alami. Dengan demikian,

pelanggan yang tidak biasa dengan notasi tersebut akan tetap dapat mengerti SKPL.

### 4.3.3 Lengkap

SKPL adalah lengkap jika dan hanya jika sudah melibatkan elemen-elemen berikut:

1. Semua kebutuhan-kebutuhan penting sudah tercakup (fungsionalitas, performansi, batasan perancangan, atribut atau antar muka eksternal). Jika ada spesifikasi lain yang telah menguraikan kebutuhan eksternal dari perangkat lunak bersangkutan, maka spesifikasi tersebut harus diacu atau dijadikan dasar (bila ada spesifikasi tambahan)
2. Definisi semua jenis masukan pada berbagai situasi, baik untuk masukan yang valid maupun tidak.
3. Referensi yang lengkap dari setiap gambar, tabel dan diagram pada SKPL, dan disertai dengan semua istilah yang digunakan dan unit yang digunakan sebagai pengukuran (bila ada).

Penggunaan istilah ‘TBD’ atau ‘To Be Defined’ atau ‘Sedang dalam pengembangan’ menunjukkan SKPL yang tidak lengkap. Tetapi bagaimana pun istilah tersebut sering harus digunakan. Pada kasus tersebut harus disertai dengan penjelasan yang menyertai pernyataan tersebut (misalnya kenapa suatu jawaban belum ditemukan), sehingga situasi tersebut dapat diselesaikan. Selain itu perlu didefinisikan cara menghilangkan pernyataan tersebut, dengan memberikan juga siapa yang bertanggung jawab, dan kapan harus sudah dihilangkan.

### 4.3.4 Konsisten

Yang dimaksud di sini adalah konsistensi internal. Jika suatu SKPL tidak mengacu ke dokumen lain yang sifatnya memiliki tingkat lebih tinggi (lebih dahulu ada, atau secara sistem lebih luas cakupannya), maka SKPL tersebut tidak benar.

Suatu dokumen tidak konsisten secara internal jika dan hanya jika tidak ada kebutuhan yang konflik. Ada tiga tipe yang dapat menyebabkan konflik dalam SKPL:

1. Karakteristik yang ditentukan pada objek nyata mungkin konflik. Misalnya:
    a. Suatu spesifikasi kebutuhan menerakan bahwa format laporan keluaran dinyatakan sebagai tabel, tetapi pada pada kebutuhan lain berbentuk tekstual.
    b. Suatu spesifikasi kebutuhan menyatakan suatu hal yang secara logis bertentangan dengan pernyataan lain
2. Kemungkinan ada konflik logika antara dua aksi, misalnya:
    a. Suatu kebutuhan mungkin menetapkan bahwa program harus menjumlah dua masukan dan kebutuhan lain mungkin menentukan harus dikali.
    b. Suatu kebutuhan menyatakan bahwa suatu status A akan mengikuti status B, tetapi pernyataan lain menyatakan status B yang akan mengikuti A.
3. Dua atau lebih kebutuhan mungkin menggambarkan atau mewakili objek dunia nyata yang sama. Namun demikian, gunakanlah istilah yang berbeda untuk menjelaskan kebutuhan yang berbeda. Misalnya suatu program yang meminta masukan dari pengguna mungkin disebut sebagai *masukan*, tetapi pada kebutuhan lain mungkin disebut *panduan* bagi pengguna.

### 4.3.5 Pengurutan berdasarkan kepentingannya/kestabilannya

Suatu SKPL diurutkan berdasarkan tingkat kepentingan/kestabilan dari setiap kebutuhannya. Hal tersebut dapat diberikan suatu tanda untuk menunjukkan kepentingan atau kestabilannya. Umumnya semua kebutuhan yang berhubungan dengan produk perangkat lunak tidak memiliki tingkat kepentingan yang sama. Beberapa kebutuhan mungkin bersifat harus, khususnya untuk aplikasi yang kritis, sementara yang lain bersifat diinginkan.

Setiap kebutuhan dalam SKPL harus diidentifikasi agar perbedaan tingkat kepentingan ini jelas dan eksplisit. Pengenalan kebutuhan yang ada dengan cara berikut mungkin akan dapat membantu:

- Pelanggan harus memberikan pemikiran yang rinci terhadap kebutuhannya. Seringkali ini akan memperjelas setiap asumsi yang sebelumnya tersembunyi.
- Pengembang harus menghasilkan keputusan/pilihan perancangan yang benar dan masing-masing memusatkan usaha dengan tingkat kesungguhan yang berbeda- beda pada bagian yang berbeda-beda dari perangkat lunak.

#### 4.3.5.1 Tingkat Kestabilan

Suatu metode untuk menyatakan kebutuhan adalah dengan menggunakan ukuran kestabilan. Kestabilan dapat dinyatakan dalam jumlah perubahan yang diharapkan pada setiap kebutuhan yang berdasarkan pada pengalaman atau pengetahuan akan kejadian yang akan datang yang mungkin dapat berakibat pada organisasi, fungsi , orang yang didukung oleh sistem perangkat lunak.

#### 4.3.5.2 Tingkat Keperluan

Tingkat keperluan dalam hal ini menggambarkan *Degree of Necessity*. Cara lain untuk mengurutkan kebutuhan adalah dengan membedakan kelas kebutuhan berdasarkan tingkat keperluan dengan menyatakannya sebagai kebutuhan yang bersifat esensial, kondisional, atau opsional.

1. Sifat **Esensial** menunjukkan bahwa perangkat lunak tidak akan diterima jika kebutuhan-kebutuhan tidak disediakan sesuai dengan persetujuan sebelumnya.
2. Sifat **Kondisional** menunjukkan bahwa kebutuhan-kebutuhan ini akan meningkatkan (*enhance*) produk perangkat lunak, tetapi produk (perangkat lunak) tetap akan diterima walaupun tidak ada.
3. Sifat **Opsional** menunjukkan kelompok fungsi yang mungkin berguna atau mungkin tidak. Pengembang akan mendapat kesempatan untuk mengajukan sesuatu yang melebihi dari apa yang tertera pada SKPL.

### 4.3.6 Dapat diverifikasi

Suatu SKPL disebut dapat diverfikasi, jika dan hanya jika setiap kebutuhan yang dinyatakan di dalamnya dapat diverifikasi. Suatu kebutuhan dapat diverifikasi jika dan hanya jika ada suatu proses yang tepat (*cost-effective*) dan dapat dilakukan

(*limited*)

untuk memeriksa apakah suatu produk perangkat lunak sudah memenuhi kebutuhan. Jadi secara umum, kebutuhan yang ambigu tidak dapat diverifikasi.

Kebutuhan yang tidak dapat diverifikasi termasuk pernyataan seperti “bekerja dengan baik”, “interaksi manusia yang baik” atau “biasanya terjadi”. Kebutuhan-kebutuhan ini tidak dapat diverikasi karena hampir tidak mungkin mendefinisikan istilah baik, atau biasanya. Pernyataan “program jangan pernah masuk ke pengulangan yang tidak terbatas (*infinite-loop*)” juga tidak dapat diverifikasi, karena pengujian kualitas ini secara teori tidak mungkin.

Contoh pernyataan yang dapat diverifikasi: “Keluaran program harus diproduksi dalam 20 detik dan harus diproduksi lagi selama 30 detik”. Statement tersebut dapat diverifikasi karena penggunaan istilah dan kuantitas yang terukur.

### 4.3.7 Dapat dimodifikasi

SKPL dapat dimodifikasi, jika dan hanya jika strukturnya memungkinkan setiap perubahan terhadap kebutuhan dapat dibuat dibuat secara mudah, lengkap dan konsisten, dengan tetap mempertahankan struktur dan gaya yang digunakan. Kemampuan dapat dimodifikasi umumnya menuntut SKPL yang,

1. memiliki organisasi yang mudah digunakan dengan daftar isi, indeks dan *cross-reference*.
2. tidak ada redundansi (duplikasi yang tidak perlu). Jadi suatu kebutuhan tidak muncul lebih dari satu tempat di SKPL
3. setiap satu kebutuhan utuh sebaiknya dinyatakan secara terpisah, yaitu tidak dicampur dengan spesifikasi kebutuhan lainnya.

Redundansi sendiri bukanlah suatu kesalahan, tetapi adalah salah satu sumber kesalahan. Redundansi dapat membantu membuat SKPL menjadi lebih mudah dibaca, tetapi akan menimbulkan masalah jika ada perbaikan terhadap dokumen yang redundan. Misalnya jika suatu kebutuhan diubah hanya pada satu dokumen sedangkan yang lainnya tidak, maka akan menimbulkan ketidak-konsistenan. Jika memang redundansi diperlukan, SKPL harus menyertakan *cross-reference* yang eksplisit yang membuatnya dapat mudah dimodifikasi.

### 4.3.8 Dapat ditelusuri

SKPL dapat ditelusuri (*traceable*) jika asal dari kebutuhan sudah jelas dan memberikan fasilitas untuk mengacu setiap kebutuhan dalam pengembangan masa depan dan perbaikan dokumen. Ada dua tipe kemampuan penelusuran yang dianjurkan:

1. Dapat ditelusuri ke belakang (*Backward Traceability*) yaitu terhadap tahapan sebelumnya. Artinya melihat pada pengacuan setiap kebutuhan ke sumber pada dokumen sebelumnya.
2. Dapat ditelusuri ke depan (*Forward Traceability*) yaitu terhadap semua dokumen yang dibuat kemudian yang didasarkan pada SKPL. Artinya melihat pada keberadaan nama atau nomor referensi yang unik setiap kebutuhan pada SKPL yang dapat diacu nantinya.

Penelusuran ke depan dari SKPL akan sangat penting ketika produk perangkat lunak memasuki fase operasi dan perawatan. Saat dokumen perancangan dan kode program di modifikasi, untuk dapat menentukan semua kebutuhan yang akan terkena dampak dari modifikasi tersebut sangat penting artinya.

## *4.4 Persiapan Bersama SKPL*

Dalam proses pengembangan perangkat lunak mula-mula harus disepakati perjanjian antara pengembang dan pelanggan tentang apa yang harus dilakukan oleh perangkat lunak. Perjanjian dalam bentuk SKPL ini harus dipersiapkan bersama. Hal ini penting karena biasanya baik pelanggan ataupun pengembang tidak dapat menulis SKPL sendiri.

- Pelanggan biasanya tidak mengerti proses perancangan dan pengembangan yang cukup untuk menulis SKPL
- Pengembang biasanya tidak mengerti masalah pelanggan untuk menentukan setiap kebutuhan.

Karena itu pelanggan dan pengembang harus bekerja sama mengembangkan SKPL.

Situasi khusus mungkin terjadi jika sistem dan perangkat lunak dibuat secara konkuren. Dalam hal ini fungsionalitas, antarmuka, performansi dan batasan lain dari perangkat lunak tidak terdefinisi sebelumnya, melainkan perlu didefinisikan secara bersama-sama, dan biasanya bergantung pada negosiasi dan dapat berubah. Kasus ini akan lebih sulit, tapi tetap harus memperhatikan hal-hal yang sudah dituliskan di 4.3. Jadi secara khusus, SKPL yang tidak sesuai dengan kebutuhan adalah SKPL yang tidak benar.

## *4.5 Evolusi SKPL*

SKPL dapat berevolusi karena kemajuan proses pengembangan perangkat lunak. Spesifikasi rinci pada saat proyek dimulai mungkin tidak bisa dilakukan. Perubahan-perubahan tambahan diperlukan untuk mengatasi defisiensi, menutupi kelemahan-kelemahan atau ketidakakuratan pada SKPL. Dua pertimbangan utama dalam proses ini adalah sebagai berikut:

1. Kebutuhan harus dispesifikasikan selengkap dan serinci mungkin walaupun tahu bahwa perubahan/revisi pasti terjadi kemudian. Ketidaklengkapan yang diketahui harus dicatat.
2. Proses perubahan formal dapat dimulai untuk mengidentifikasi, mengendalikan, merunut dan melaporkan setiap perubahan. Perubahan kebutuhan yang disetujui disertakan pada SKPL untuk:
   - menyediakan jejak audit yang lengkap dan akurat dari perubahan
   - memungkinkan pengkajiulangan bagian SKPL yang kini maupun yang akan datang

## *4.6 Prototipe*

Prototipe atau purwarupa sering digunakan selama fase kebutuhan dalam proyek. Banyak kakas yang sudah tersedia untuk membuat prototipe, yang hanya

menampilkan beberapa aspek karakteristik dari sistem. Prototipe ini biasanya dibuat dengan sangat cepat dan mudah. Prototipe berguna karena tiga alasan:

1. pelanggan biasanya lebih mudah melihat prototipe dan bereaksi daripada membaca SKPL. Jadi kadang-kadang prototipe memungkinkan pengembang mendapatkan umpan balik dari pelanggan dengan cepat
2. Prototipe mampu menampilkan aspek yang tidak terantisipasi pada perilaku sistem. Jadi kadang-kadang tidak hanya memberikan jawaban tetapi juga menimbulkan pertanyaan baru. Hal ini akan membantu mengungkapkan semua kebutuhan yang perlu ada pada SKPL
3. SKPL yang menggunakan teknik prototipe cenderung lebih sedikit mengalami perubahan selama pengembangan, sehingga dapat mengurangi waktu pengembangan.

## *4.7 Pemasukan Rancangan pada SKPL*

Suatu kebutuhan mendefinisikan fungsi pada sistem yang kelihatan dari luar. Suatu rancangan menjelaskan tentang subkomponen dari sistem atau antarmukanya dengan komponen lain. Penulis SKPL harus secara jelas membedakan antara memberikan **batasan rancangan** (*design constraints*) dengan **rancangan** (*design*) itu sendiri. Setiap kebutuhan dalam SKPL akan membatasi alternatif rancangan yang bisa dipilih. Tetapi hal ini bukan berarti setiap kebutuhan adalah rancangan.

Pada kasus khusus, beberapa kebutuhan akan membatasi rancangan secara tegas. Misalnya kebutuhan akan keamanan atau keselamatan mungkin dapat dirancang secara langsung ke perancangan, misalnya untuk keperluan:

- Pemisahan fungsi dan modul
- Ijin komunikasi terbatas antar beberapa area dalam program
- Pemeriksaan integritas data untuk variabel kritis

Contoh batasan perancangan yang baik adalah kebutuhan fisik, kebutuhan performansi, standard pengembangan perangkat lunak dan standard pemastian kualitas perangkat lunak.

Jadi kebutuhan harus dinyatakan murni dari sudut pandang eksternal. Jika menggunakan suatu model untuk mengilustrasikan kebutuhan, ingat bahwa model hanya menyatakan perilaku eksternal, dan tidak menspesifikasikan rancangan.

## *4.8 Pemasukan kebutuhan proyek pada SKPL*

SKPL seharusnya mengarah ke produk perangkat lunak, bukan proses untuk menghasilkannya. Kebutuhan projek menyatakan persetujuan antara pelanggan dan pengembang tentang masalah kontrak yang berhubungan dengan produksi perangkat lunak dan sebaiknya ***tidak diikut sertakan dalam SKPL***. Hal-hal yang menyangkut kebutuhan projek antara lain:

1. Biaya
2. Jadwal penyerahan
3. Aturan pelaporan
4. Metode Pengembangan Perangkat Lunak

5. Jaminan Kualitas
6. Kriteria Validasi dan Verifikasi
7. Aturan penerimaan (*acceptance procedure*).

Kebutuhan proyek akan ditentukan pada dokumen lain, umumnya dalam dokumen perencanaan pengembangan perangkat lunak, perencanaan jaminan kualitas atau *statement of work*.

## 5. Bagian-bagian SKPL

Isi SKPL secara umum bergantung pada pendekatan yang dilakukan untuk mempresentasikan hasil analisis (berorientasi proses, objek, atau perilaku), karena uraian penjelasan hasil analisis bergantung pada model analisis yang dihasilkan. Kerangka umum SKPL memuat hal-hal sebagai berikut:

1. Pendahuluan
    a. Tujuan
    b. Lingkup
    c. Definisi dan Singkatan (Akronim)
    d. Referensi
    e. Deskripsi Umum Dokumen
2. Deskripsi Umum Perangkat Lunak
    a. Deskripsi Umum Sistem
    b. Fungsi Produk
    c. Karakteristik Pengguna
    d. Batasan-batasan
    e. Lingkungan Operasi
3. Deskripsi Rinci Kebutuhan

Untuk setiap sistem (kecuali yang sangat sederhana) kebutuhan rinci cenderung menjadi luas. Oleh karena itu, direkomendasikan dua cara untuk mengorganisasikan deskripsi rinci kebutuhan yaitu untuk pendekatan berorientasi proses dan untuk pendekatan berorientasi objek. Penjelasan tentang Deskripsi Rinci Kebutuhan diulas pada Panduan GL01A untuk pendekatan berorientasi proses dan Panduan GL01B untuk pendekatan berorientasi objek.

Banyak notasi, metode, teknik dan dukungan kakas otomatis untuk membantu dokumentasi kebutuhan. Umumnya penggunaannya bergantung pada pendekatan yang digunakan. Ada banyak pendekatan yang bisa dipakai selain kedua pendekatan tersebut di atas. Setiap pendekatan mengarahkan pengorganisasian SKPL yang berbeda-beda. Beberapa dari organisasi selain kedua pendekatan (organisasi) di atas dijelaskan sebagai berikut:

- Mode sistem

  Beberapa sistem berlaku agak berbeda tergantung pada modus operasi sistem. Sebagai contoh sistem kendali mungkin memiliki sekumpulan fungsi yang berbeda tergantung modenya (training, normal atau berbahaya).

- Kelompok pengguna

  Beberapa sistem membutuhkan fungsi yang berbeda terhadap kelompok yang berbeda dari pengguna. Sebagai contoh sistem elevator melibatkan masyarakat

umum sebagai pengguna elevator, pekerjan maintenance dan pemadam kebakaran.

- *Feature*

  Suatu feature adalah pelayanan yang dinginkan secara eksternal oleh sistem yang membutuhkan serangkaian masukan yang memberi efek terhadap hasil. Sebagai contoh pada sistem telpon, feature-nya adalah hubungan lokal, *call forwarding* dan *conference call.* Setiap feature umumnya dijelaskan dalam pasangan stimulus- response.

- Stimulus

  Beberapa sistem akan lebih baik diorganisasikan berdasarkan stimulus. Misalnya fungsi-fungsi sistem pendaratan pesawat udar mungkin diorganisasikan menjadi bagian *loss of power, wind shear, sudden change in roll, vertical velocity excessive,* dll.

- Respons

  Beberapa sistem dapat diorganisasikan dengan menjelaskan semua fungsi dalam mendukung pembangkitan respons. Misalnya fungsi sistem personil dapat diorganisasikan menjadi bagian-bagian yang berhubungan dengan semua fungsi yang diasosiasikan dengan pembangkitan cek pembayaran, fungsi-fungsi yang berhubungan daftar pegawai, dan lain-lain.

Contoh dari pengorganisasi alternatif di atas dapat dilihat pada IEEE Std. 830-1993.