BAB II LANDASAN KEPUSTAKAAN

1.1 Tinjauan Pustaka

Pada penelitian pembagunan sistem informasi kenaikan jabatan fungsional Universitas Brawijaya ini memiliki aspek utama dalam hal pemenuhan proses bisnis dimana sistem informasi yang dikembangkan dibangun sesuai proses bisnis serta membantu mengautomatisasi proses bisnis yang ada. Terdapat beberapa penelitian terhadap permasalahan yang mengangkat dua aspek utama tersebut serta penelitan terdahulu yang serupa.

Setyadi (2013) melakukan penelitian dengan judul sistem informasi penilaian angka kredit dosen, dibuat sistem menggunakan Visual Basic. Penelitian ini merupakan penelitian implementatif yang membahas masalah dibutuhkan waktu dan tenaga yang lebih untuk mengingatkan dosen yang lama tidak naik jabatan, dikarenakan petugas kepegawaian harus mencari berkas setiap dosen yang harus diberi peringatan tersebut serta dosen tida terarah dalam pemenuhan unsur-unsur yang menjadi syarat kenaikan jabatan fungsional. Penelitian ini menggunakan dua teknik metodologi penelitian, yaitu teknik pengumpulan data dan teknik pengembangan sistem. Dalam sistem yang dibuat, sistem membantu dosen dalam mencatat kegiatan yang dimasukkan dalam penilaian angka kredit dan mengetahui nilai dari masing-masing kriteria serta prosentasenya.

Penelitian yang dilakukan Przybyłek (2013) yang berjudul *bridging the gap between business proses model and use-case models* melakukan penelitian terhadap cara menghubungkan model proses bisnis ke model *use case*. Tujuan dari penelitian ini adalah mengembangkan sebuah pendekatan baru yang mendukung elisitasi kebutuhan disaat kebutuhan klien tidak pasti dan membuat kebutuhan sistem benar-benar mendukung proses bisnis yang mendasarinya. Dalam pemodelan proses bisnis di penelitian ini menggunakan UML *activity diagram*. Pendekatan ini memiliki 3 tahapan, pertama memodelkan proses bisnis saat ini yang akan menghasilkan output model *As-Is*. Kedua meningkatkan proses bisnis bertujuan untuk menentukan proses yang akan diautomatisasi, dan ketiga elisitasi kebutuhan fungsional yang menghasilkan model *To-Be*, *use case diagram*, serta *traceability matrix* untuk mencocokkan apakah *use case* yang ada mencakup proses pada proses bisnis. Hasil dari penelitian ini menunjukkan bahwa dengan 3 pendekatan tersebut dapat memperoleh kesinambungan dalam pemodelan proses bisnis dengan kebutuhan fungsional sistem yang dalam hal ini digambarkan dengan use case diagram.

Cruz, *et al* (2014) dalam penelitiannya yang berjudul *from business process models to use case models : a systematic approach* melakukan penelitian bagaimana membangun model use case berdasarkan pemodelan proses bisnis. Dalam penelitian ini pemodelan proses bisnis menggunakan BPMN. Cruz, *et al* mengatakan bahwa tidak semua informasi di BPMN dapat diubah menjadi sebuah aktor atau sebuah use case. Informasi tersebut akan dicantumkan dalam deskripsi *use case*. Pendekatan ini memiliki 2 tahap. Pertama akan menyajikan sekumpukan *rule* atau aturan untuk memeperoleh *use case* diagram berdasarkan model BPMN. kedua menggunakan *rule* tadi untuk memperoleh deskripsi dari *use case* yang sebelumnya telah diidentifikasi. Hasil dari 2 pendekatan tersebut dapat menghubungkan BPMN sebagai dasar untuk memperoleh kebutuhan fungsional yang dimodelkan dengan use case diagram.

2.2 Dasar Teori

Didalam dasar teori ini menjelaskan hal-hal yang berkaitan dengan perancangan sistem informasi kenaikan pangkat dan jabatan fungsional dosen.

2.2.1 Pangkat dan Jabatan Fungsional Dosen

Berdasarkan Peraturan Menteri Pendayagunaan Aparatur Negara dan Reformasi Birokrasi Nomor 17 Tahun 2013, Bab 1 Ketentuan Umum, Pasal 1, menyebutkan “Jabatan fungsional dosen selanjutnya disebut jabatan akademik dosen adalah kedudukan yang menunjukkan tugas, tanggung jawab, wewenang dan hak seseorang dosen dalam suatu satuan pendidikan tinggi yang dalam pelaksanaannya didasarkan pada keahlian tertentu serta bersifat mandiri”. Pada Buku Pedoman Operasional Penilaian Angka Kredit Kenaikan Pangkat/Jabatan Akademik Dosen, Direktur Jendral Pendidikan Tinggi pada saat itu, Djoko Santoso mengatakan, “jabatan fungsional dosen pada dasarnya merupakan pengakuan, penghargaan dan kepercayaan atas kompetensi, kinerja, integritas dan tanggung jawab dalam pelaksanaan tugas, serta tata karma dosen dalam melaksanakan tugas tridarmanya.”

Dengan kedua pernyataan diatas dapat disimpulkan pengertian pangkat dan jabatan fungsional dosen adalah suatu kedudukan sebagai penghargaan yang diperoleh atas prestasi kinerja selama menjadi dosen dan merupakan hak dan wewenang bagi setiap dosen. Pangkat dan jabatan ini diharapkan juga dapat berfungsi sebagai insentif non materi bagi dosen untuk lebih bekerja keras, lebih kreatif, dan lebih baik dalam segala aspek.

Adapun jenjang pangkat dan jabatan fungsional dosen dapat dilihat pada Tabel 2.1.

Tabel 2.1 Jenjang pangkat dan jabatan fungsional dosen

No	Jabatan Fungsional	Golongan	Angka Kredit Kumulatif
a	Asisten Ahli	III/b	150
b	Lektor	III/c	200
		III/d	300
c	Lektor Kepala	IV/a	400
		IV/b	550
		IV/c	700
d	Guru Besar/Professor	IV/d	850
		IV/e	1050

Sumber : (Peraturan MenPAN dan RB No. 17 tahun 2013)

Angka kredit kumulatif merupakan jumlah nilai yang harus diperoleh oleh dosen melalui aktivitas unsur utama yaitu tridharma perguruan tinggi meliputi pendidikan (meliputi pendidikan sekolah dan pelaksanaan pendidikan/pengajaran), penelitian (meliputi penghasilan karya ilmiah dan pelaksanaan penelitian), serta pengabdian kepada masyarakat, dan unsur penunjang yang merupakan aktivitas pendukung pelaksanaan tugas pokok dosen yang memeiliki nilai angka kredit pada setiap komponennya. Berdasarkan Peraturan Menteri Pendayagunaan Aparatur Negara dan Reformasi Birokrasi Nomor 17 Tahun 2013, Angka Kredit adalah satuan nilai dari tiap butir kegiatan dan/atau akumulasi nilai butir-butir kegiatan yang harus dicapai oleh seorang Dosen dalam rangka pembinaan karier kepangkatan dan jabatan. Angka kredit kumulatif inilah yang harus dipenuhi oleh dosen untuk dapat menduduki jenjang pangkat dan jabatan fungsional tertentu.

2.2.2 Sistem Informasi

Sistem Informasi mencakup sejumlah komponen (manusia, computer, teknologi informasi dan prosedur kerja), ada sesuatu yang diproses(data menjadi informasi), dan dimaksudkan untuk mencapai suatu sasaran atau tujuan (Kadir, 2014). Dari pengertian tersebut dapat disimpulkan bahwa sistem informasi merupakan sekumpulan kerangka kerja yang bekerja sama dalam memproses suatu *input*an data yang diproses menjadi output berupa informasi untuk mencapai suatu sasaran atau tujuan baik dalam suatu perusahaan maupun dalam suatu organisasi.

2.2.3 Proses Bisnis

Proses bisnis adalah sekumpulan kegiatan yang membutuhkan satu atau lebih jenis *input* dan akan menghasilkan suatu keluaran tertentu seperti laporan atau peramalan yang berguna bagi pelanggan (Monk & Wager, 2013). Pelanggan disini dalam proses bisnis dapat berupa pelanggan eksternal tradisional, seseorang yang membeli produk jadi, dan pelanggan internal seperti rekan kerja di lain departemen. Dengan kata lain, proses bisnis memiliki pengertian sekumpulan dari banyak aktivitas atau pekerjaan yang terstruktur dan saling berhubungan untuk menyelesaikan suatu masalah tertentu atau menghasilkan suatu keluaran, dan membantu pencapaian tujuan serta sasaran strategis dari suatu organisasi. Secara umum organisasi memiliki tujuan agar dapat berjalan pada jenjang waktu yang lama dan selalu meningkatkan produktifitasnya. Proses bisnis yang baik dibutuhkan oleh suatu organisasi untuk mendukung berjalannya organisasi tersebut sehingga tujuan organisasi dapat dicapai. Kegiatan proses bisnis ini dapat dilakukan secara manual ataupun dengan menggunakan bantuan sistem informasi.

2.2.3.1 *Business Process Model and Notation* (BPMN)

Business Process Model and Notation dikembangkan oleh the *Business Process Management Initiative* (BPMI). *Business Process Modeling and Notation* (BPMN) adalah standar untuk pemodelan proses bisnis yang menyediakan notasi grafis untuk menentukan proses bisnis dalam Proses Bisnis Diagram (BPD) dengan menggunakan teknik diagram alir tradisional (van Rosing et al , 2015). Tujuan dari BPMN adalah untuk membantu pemodelan proses bisnis, baik untuk pengguna teknis, maupun pengguna bisnis, dengan menyediakan notasi intuitif untuk pengguna bisnis, namun dapat mewakili proses semnatik yang kompleks.

BPMN dirancang dan digunakan oleh semua pemangku kepentingan dengan konsep yang mudah dipahami, mulai dari bisnis analis yang bertugas menciptakan konsep awal dari sebuah proses bisnis, para pengembang teknis yang memiliki tanggung jawab untuk mengimplementasikan teknologi yang mengelola proses-proses tersebut, serta manajer bisnis yang memantau dan mengelola proses tersebut. Maka dari itu BPMN menjadi bahasa umum yang menjembatani kesenjangan komunikasi yang terjadi antara desain proses bisnis dan implementasi.

BPD terdiri dari satu set elemen notasi grafis yang memiliki unsur-unsur yang memungkinkan pengembangan diagram sederhana lebih mudah dan akan tampak familiar untuk sebagian besar bisnis analis, contoh diagram sederhana adalah *flowchart diagram*. Unsur-unsur yang yang ada dalam BPD dibedakan satu dengan yang lain agar pemodel bisnis mudah memahaminya, misalnya *activity* berbentuk segitiga dan *decisions* berbentuk belah ketupat. Yang perlu digaris bawahi adalah

salah satu tujuan dalam pengembangan BPMN adalah menciptakan mekanisme proses yang sederhana dalm pembuatan pemodelan bisnis proses, namun pada saat yang sama BPMN harus mampu menangani kompleksitas yang melekat pada proses bisnis. Pendekatan yang dilakukan untuk menangani dua persyaratan yang bertentangan ini adalah dengan mengatur aspek grafis ke dalam kategori tertentu. BPD menyediakan satu set kecil kategori notasi sehingga pembaca BPD dapat dengan mudah mengenali tipe dasar elemen dan memahami diagram. Terdapat empat kategori dasar elemen.

1. *Flow Object*

BPD memiliki tiga elemen utama yang termasuk dalam *flow object*, sehingga para pemodel tidak harus belajar dan mengenali banyak bentuk yang berbeda-beda. Ketiga elemen *flow object* dapat dilihat pada Tabel 2.2

Tabel 2.2 Elemen utama *flow object*

No	Nama	Keterangan	Gambar
1	*Event*	*Event* digambarkan dengan lingkaran yang merupakan suatu simbol yang menggambarkan apa yang sedang terjadi selama berlangsungnya proses bisnis. *Event* ini yang memiliki pengaruh dalam *Flow Object* yang biasanya memiliki penyebab (*trigger*) atau dampak (*result*). *Event* digambarkan dengan lingkaran pusat terbuka memiliki arti sebagai penanda internal yang akan membedakan *trigger* atau *result*. Terdapat 3 jenis *event* berdasarkan fungsi dalam *flow*, yaitu *Start*, *Intermediate*, dan *End*	
2	*Activity*	Sebuah *activity* digambarkan persegi panjang yang melengkung disudutnya dan sebagai simbol yang menggambarkan pekerjaan umum pada sebuah perusahaan. Terdapat 2 jenis *activities*, yaitu *Task* dan *Sub-Process*.	
3	*Gateway*	*Gateway* digambarkan dengan gambar belah ketupat yang memiliki fungsi sebagai pengendali percabangan dan peggabungan dari	

No	Nama	Keterangan	Gambar
		sequence fLow. Maka dari itu *Gateway* digunakan dalam pengambilan keputusan, seperti percabangan dan penggabungan.	

2. *Connecting Object*

Connecting Object merupakan elemen yang menghubungkan elemen *Flow Object* dalam diagram. Terdapat tiga jenis elemen dalam C*onnecting Object* seperti pada tabel 2.3.

Tabel 2.3 Elemen dari *connecting object*

No	Nama	Keterangan	Gambar
1	*Sequence Flow*	*Sequence FLow* diggambarkan dalam bentuk garis dengan tanda panah yang tegas atau solid yang berfungsi untuk menunjukkan urutan kegiatan yang akan terjadi dalam proses.	
2	*Message Flow*	*Message Flow* digambarkan dengan garis putus-putus dan tanda panah terbuka yang berfungsi untuk menunjukkan aliran pesan antara dua entitas yang terpisah yang digunakan untuk mengirim dan menerima pesan.	
3	*Association*	*Association* digambarkan dengan garis titik dengan panah yang berfungsi sebagai asosiasi data, teks, dan *Artifact* lainnya dengan *Flow Object*. *Association* digunakan untuk menunjukkan masukan dan keluaran dari suatu proses	

3. *SwimLanes*

SwimLane banyak digunakan dalam metodologi pemodelan proses sebagai mekanisme dalam mengatur kegiatan dalam kategori visual yang

terpisah untuk menggambarkan kemampuan fungsional dan tanggung jawab yang berbeda. Di dalam BPMN terdapat dua kategori *SwimLane.*

Gambar 2.1 Gambar *SwimLane*

Kategori pertama adalah *Pool*. *Pool* berfungsi sebagai wilayah bagi semua partisipan yang terdapat dalam proses. Setiap *pool* dapat berupa suatu organisasi yang konkrit, atau dapat berupa *placholder* untuk sebuah organisasi tertentu yang memiliki peran tersendiri dalam suatu proses, misal organisasi yang memiliki peran supplier, produsen, serta customer. Setiap proses berada pada satu *Pool*, dengan demikian konsekuensinya adalah setiap satu proses dilakukan oleh satu organisasi. Proses bisnis dapat beriteraksi dengan proses bisnis yang terdapat atau ditetapkan oleh organisasi lainnya sehingga tercipta proses B2B atau *business to business*, yaitu proses bisnis yang melibatkan proses bisnis lain yang berbeda organisasi.

Kategori selanjutnya dalam *SwimLane* adalah *Lane. Lane* merupakan *Sub-Partition* dari *Pool* yang akan memperpanjang ukuran dari *Pool,* baik secara vertikal maupun horizontal. *Lane* sangat melekat dengan proses *SwimLane* dalam metode pemodelan tradisional. *Lane* sering digunakan untuk memisahkan kegiatan yang berhubungan dengan fungsi atau peran perusahaan tertentu. *Sequence flow* dapat menyebrangi batas-batas *Lanes* dalam *Pool,* tetapi *message flow* tidak dapat digunakan antara *fLow objects* dalam *Lanes* pada *Pool* yang sama.

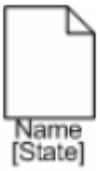
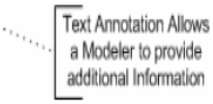
4. Artefak

BPMN dirancang untuk memungkinkan pemodel dan *tool* untuk pemodel lebih fleksibel dalam memperluas notasi dasar dan dalam memberikan kemampuan untuk konteks tambahan yang sesuai untuk situasi pemodelan tertentu, seperti untuk market vertikal misalnya, asuransi atau perbankan. Artefak dapat ditambahkan ke diagram yang sesuai dengan konteks proses bisnis yang dimodelkan.

Artefak digunakan untuk menampilkan informasi tambahan tentang proses bisnis yang tidak secara langsung relevan dengan proses *Sequence Flow* atau *Message Flow.* Setiap Artefak dapat dikaitkan dengan unsur-unsur alur proses diagram. Tujuan dari Artefak adalah menampilkan informasi, sehingga

eksekusi proses semantik tidak dipengaruhi oleh keberadaan artefak. Elemen yang mendukung Artefact dapat dilihat pada Tabel 2.4

Tabel 2.4 Elemen BPD dari artefak

No	Nama	Keterangan	Gambar
1	*Data Object*	*Data Object* merupakan mekanisme untuk menunjukkan bagaimana data dibutuhkan atau dihasilkan oleh aktivitas. Tujuan utama dari *Data Object* adalah sebagai dokumentasi penggunaan data dalam proses.	
2	*Group*	*Group* digambarkan dengan persegi dengan sudut melengkung yang digambar dengan garis putus-putus. *Group* dapat digunakan untuk dokumentasi atau menganalis tujuan, tetapi tidak mempengaruhi *Sequence Flow*.	
3	*Annotation*	*Annotation* merupakan *tool* untuk pemodel untuk menyediakan tambahan informasi teks bagi pembaca diagram BPMN	

2.2.3.2 Pemetaan bisnis proses menjadi spesifikasi kebutuhan

Dua kebutuhan utama dalam mengembangkan sebuah sistem informasi untuk sebuah organisasi adalah sistem analis harus mengetahui dan memahami kinerja dari sistem informasi yang dikembangkan dan sistem yang dikembangkan benar-benar mendukung proses bisnis dari organisasi itu (González, 2011). Namun nyatanya terdapat dua hambatan untuk menyelaraskan antara proses bisnis dan sistem informasi yang dikembangkan. Pertama sistem informasi tidak dibangun dengan pemahaman proses yang yang tepat. Kedua bisnis proses tidak memiliki hubungan dalam kebutuhan sistem sehingga bersifat terpisah atau independen dari sistem informasi yang dikembangkan (Przybyłek, 2013).

Menurut Przybyłek (2013) terdapat tiga tahap pendekatan dalam mengatasi masalah tersebut:

1. Pemodelan bisnis proses *As-Is*

Tahap ini bertujuan untuk memahami organisasi dengan melihat dimana sistem informasi akan dibangun. Pemodelan proses bisnis memiliki peran yang utama dalam tahapan ini. Untuk memodelkan proses bisnis dibutuhkan pengetahuan terhadap aktifitas organisasi dan lingkungan bisnis dari organisasi. Pengetahuan ini dapat didapatkan dengan melakukan observasi dan interview dengan pegawai yang berada dalam wilayah eksekusi proses bisnis. Selanjutnya dilakukan pemodelan proses bisnis berdasarkan informasi yang didapat. Biasanya dilakukan sejumlah iterasi untuk mendapatkan versi akhirnya.

2. Peningkatan proses bisnis

Setelah proses bisnis dimodelkan, langkah selanjutnya adalah kesepakatan tentang bagian bisnis mana yang akan di automatisasi. Dua faktor penting yang utama adalah harus memperhatikan biaya dan manfaat. Pada tahap ini harus mengajak pemangku kepentingan ikut berperan dalam memberikan umpan balik tentang hasil pemodelan. Tahap ini akan menghasilkan konsep model proses bisnis *To-Be* dan daftar proses yang akan dikomputerisasikan.

3. Elisitasi kebutuhan fungsional

Sangat penting untuk memastikan bahwa sistem informasi yang dikembangkan akan menyelesaikan kebutuhan sesungguhnya dari bisnis itu. Oleh karena itu setiap use case yang diusulkan untuk sistem harus memiliki asal usul yang jelas setidaknya satu proses bisnis. Lalu untuk menelusuri kecocokan use case dengan menggunakan teknik *traceability matrix*. Pada Table 2.5 merupakan contoh *Traceability matrix* biasanya diimplementasikan sebagai tabel atau spreadsheet. Proses diasosiasikan sebagai baris dan use case diasosiasikan sebagai kolom matriks. Ketika sebuah proses berhubungan dengan use case, sebuah tanda akan ditempatkan di sel yang berpotongan. Hasil dari tahap ini akan menghasilkan model proses bisnis *To-Be*, *use case* diagram, dan *traceability matrix.*

Tabel 2.5 Contoh *traceability matrix*

	enroll	withdraw application	take exam	view status	review application	create exam	activate exam	adjust admission setting	create student list
enrolling in the system	x	x		x					
submitting document	x								
validating application					x				
informing about exam								x	
preparing exam						x			
conductiong exam			x						
reviewing exam						x			
calculating the matriculation score					x			x	
drawing up a list of initially accepted									x
sending acceptance / rejection letter									x
drawing up a list of first year students									x
paying a recruitment fee									
paying all tuition fees									
sending original diplomas									

Sumber : (Przybyłek, 2013)

2.2.4 Sistem Development Life Cycles (SDLC)

Sistem Development Life Cycles (SDLC) adalah proses menentukan bagaimana sistem informasi (SI) dapat mendukung kebutuhan bisnis, merancang sistem, membangun sistem, serta menyerahkan sistem kepada pengguna (Dennis et al, 2012). Menurut Joko Dewanto (2004), *Sistem Development Life Cycles* (SDLC) merupakan suatu urutan dari beberapa proses yang dilakukan secara bertahap didalam merancang serta mengembangkan sistem yang dikenal sebutan *Information Sistem Development* atau juga *Application Development.* Prinsip siklus digunakan dalam pengembangan sistem pada saat ini karena sistem akan selalu memerlukan suatu perubahan, seperti perkembangan jaman, perkembangan teknologi, pelanggan dan lainnya. Sehingga siklus atau putaran suatu tahapan pengembangan sistem akan dimulai dari awal lagi.

SDLC memiliki tiga tujuan utama, yaitu memastikan sistem yang dikembangkan berkualitas tinggi, memberikan kontrol manajemen yang kuat atas proyek, dan memaksimalkan produktivitas dari staff sistem (Bender, 2003). Dalam rangka untuk memenuhi tujuan SDLC tersebut harus memenuhi beberapa persyaratan tertentu, yaitu harus mampu mendukung proyek-proyek dan sistem dari berbagai lingkup dan jenis, mendukung semua kegiatan teknis, mendukung semua kegiatan manajemen,

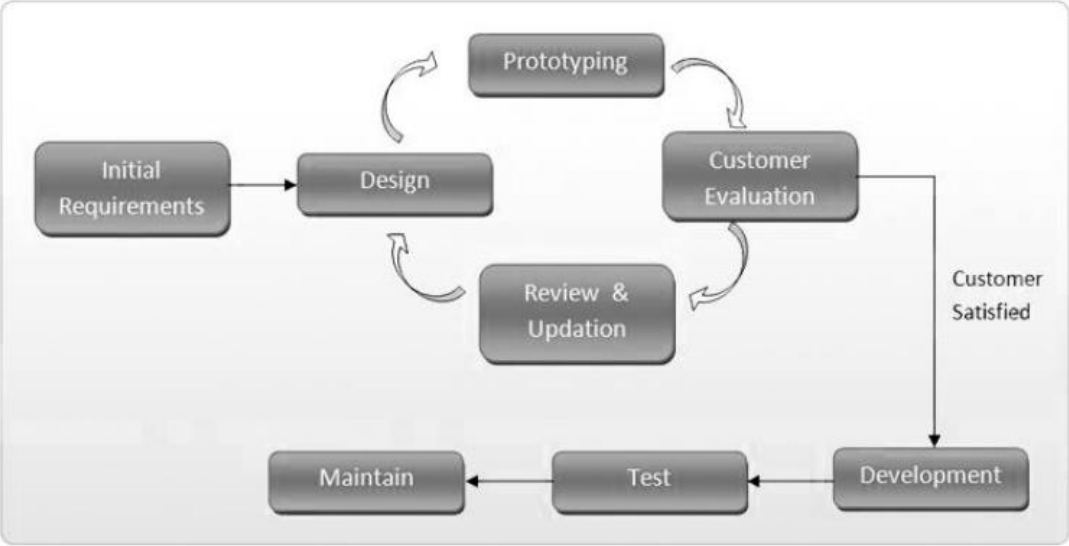
serta menyediakan panduan tentang bagaimana cara mengisntal sistem yang dikembangkan.

SDCL merupakan suatu kerangka kerja atau disebut *framework* yang terstruktur dan berisi proses-proses terurut bagaimana pengembangan sistem informasi tersebut berjalan. Didalam SDLC terdapat beberapa model metode, salah satunya adalah motode model *Prototyping* dimana peneliti akan memakai metode model *Prototyping* ini dalam pembangunan sistem informasi kenaikan jabatan fungsional dosen di Universitas Brawijaya.

2.2.4.1 Model *Prototyping*

Model *prototyping* merupakan suatu teknik untuk mengumpulkan informasi tertentu mengenai kebutuhan-kebutuhan informasi pengguna secara cepat. Berfokus pada penyajian aspek-aspek perangkat lunak tersebut yang akan nampak bagi pelanggan atau yang akan nampak bagi pemakai. *Protoype* tersebut akan dievaluasi oleh pelanggan/pemakai dan diapaki untuk menyaring kebutuhan pengembangan perangkat lunak. Model *prototyping* merupakan proses *iterative* dalam pengembangan sistem dimana kebutuhan/*requirement* diubah ke dalam sistem yang bekerja (*working sistem)* yang secara terus menerus diperbaiki melalui kerjasama antara user dan analis (Al fatta, 2007). Menurut Ogedebe p dan Jacob B (2012), *prototype* memungkinkan pengembang perangkat lunak untuk mendapatkan informasi tentang kebutuhan pengguna dengan cara memperkenankan pengguna untuk berinteraksi dengan *prototype*.

Sebuah *protoype* merupakan *sample* dari implementasi sistem yang menunjukkan batasan dan kapabilitas fungsional utama dari sistem yang diusulkan. Setelah *prototype* dibangun, akan diserahkan kepada *user/*pengguna untuk dievaluasi. *Prototype* membantu *user/*pengguna dalam menentukan bagaimana fitur akan berfungsi pada sistem final. Pengguna akan memberikan saran dan improvisasi untuk *prototype*-nya.



Gambar 2.2 Model prototyping

Sumber : (Tanwar, 2016)

Tim pengembang mengimplementasikan saran yang diberikan oleh pengguna kedalam *prototype* yang baru, nantinya akan dievaluasi kembali oleh *user.* Prosess berlanjut sampai *user* dan tim pengembang saling memahami kebutuhan sesungguhnya dari sistem yang dikembangkan. Ketika *final prototype* telah dikembangkan, kebutuhan atau *requirement* dianggap telah benar dan ditetapkan (Tanwar, 2016). Alur pada model *prototyping* pada umumnya digambarkan seperti pada Gambar 2.2.

2.2.4.2 Keuntungan dan Kelebihan model Prototype

Keuntungan dari model *protoype* sangat banyak, model ini dapat digunakan sebagai alat komunikasi anatara pengembang dan pengguna untuk mengatasi masalah yang telat didefinisikan dari perangkat lunak, *over-budgeting* dan menghasilkan produk yang sedikit memenuhi kebutuhan fungsional yang diinginkan. Semua itu dilakukan dengan tindakan manajemen kebutuhan. *Prototyping* juga dapat membantu untuk mengatasi masalah dari sedikitnya masukan dari pengguna, definisi kebutuhan yang tidak lengkap serta perubahan kebutuhan.

Prototype membantu meringankan pengembang untuk melakukan pembenahan dan perubahan dalam software yang dikembangkan, hal ini biasanya berdasarkan umpan balik dari pengguna (M Ogedebe dan Babatunde, 2012). Keunggulan lainnya menut Kumar Gupta (2015) adalah pengguna secara aktif terlibat dalam pengembangan perangkat lunak, pengguna lebih dapat memahami bagaimana

perangkat lunak dikembangkan, dapat mendeteksi kesalahan lebih awal, kebingungan dan kesulitan dalam fungsi dapat teridentifikasi.

Meskipun banyak keunggulan dan manfaat dari model *prototyping,* banyak juga kelemahan dan potensial resiko yang terdapat pada model *prototyping*. Kelemahan utama dan paling utama adalah faktanya pengguna akan berulang kali bertanya tentang perubahan dalam perangkat lunak yang dikembangkan. Hal ini dikarenakan pengguna akan melakukan pengecekan ulang *protoype* yang ada dan cenderung untuk mencatat fitur-fitur baru yang mereka inginkan untuk dimasukkan kedalam produk akhir (Ogedebe p dan Jacob B, 2012). Kelemahan lainnya menurut Kumar Gupta (2015) adalah memungkinkan sistem dibiarkan sebelum selesai, memungkinkan sistem diimplementasi sebelum siap.

2.1.5 Unified Modeling Language (UML)

Pada tahap *analys* dan *design*, pada penelitian ini akan menggunakan UML dalam pemodelan sistem. *Unified Modeling Language* (UML) adalah tujuan umum bahasa pemodelan yang digunakan untuk menentukan, memvisualisasi, membangun dan mendokumentasikan artefak dari sistem perangkat lunak (Rumbaugh et al, 2005). UML adalah bahasa pemodelan yang didefinisikan dengan baik dan merupakan sintaks yang tepat dan semantik yang dapat diinterpretasikan dan diubah oleh komputer (A. Ansari, 2012). Penggunaan UML dalam pemodelan sistem dilakukan pada tahap analisis dan design dari sebuah siklus pengembangan perangkat lunak.

Modeling adalah bagian yang sangat penting untuk memahami masalah *real-time.* Model UML menerima gambaran nyata dari sistem yang sebenarnya dan menggambarkan dalam bentuk gambar dan notasi. UML menawarkan sebuah standar dalam merancang suatu model dari sebuah sistem. Dengan menggunakan UML dapat dibuat model untuk semua jenis aplikasi perangkat lunak, dimana aplikasi tersebut akan berjalan pada perangkat keras, sistem operasi dan jeringan apapun, serta ditulis dalam bahasa pemograman apapun. UML bukan merupakan bahasa pemograman utama, UML dapat digunakan untuk menulis program, tetapi tidak memiliki kemudahan sintaksis dan semantis pada kebanyakan bahasa pemograman yang menyediakan kemudahan dalam hal pemograman. Dengan bantuan alat atau *tools* dapat menghasilkan kode dari UML ke berbagai bahasa pemograman, serta membangun konsep *reverse-engineered* dari program yang ada. Dengan menggunakan UML, fokus dari suatu rekaya perangkat lunak akan terjadi pada tahap pengembangan dan desain, diantaranya: tinjauan secara umum mengenani arsitektur dari sistem secara keseluruhan, memahami bagaimana proses interaksi antara suatu objek didalam sistem dapat saling berkirim pesan (*message)* dan saling berhubungan antara satu dengan lainnya, melakukan testing atau pengujian sistem dengan tujuan

apakah sistem telah bekerja sesuai yang diharapkan, mendokumentasikan pengembangan sistem untuk kebutuhan di masa yang akan datang.

UML memiliki banyak *tool* yang mendukung pemodelan sistem, *tool* yang digunakan dalam penelitian ini dalam melakukan pemodelan sistem diantaranya adalah *use case diagram, activity diagram, sequence diagram,* dan *class diagram*. Pada sub-bab berikut ini akan dijelaskan mengenai setiap tahapan dalam proses pada metode model *prototype* dan *tools* UML apa saja yang akan digunakan nantinya dalam mendukung pembangunan perangkat lunak pada setiap fase model *prototype*.

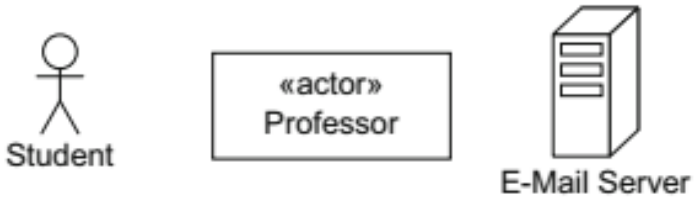
2.2.5.1 Use-case diagram

UML merekomendasikan menggunakan *use case diagram* kepada pengembang perangkat lunak dalam menentukan persyaratan apa saja yang harus dipenuhi oleh sistem. Diagram ini menggambarkan pengguna yang menggunakan fungsi dari sistem tetapi tidak membahas rincian spesifikasi kinerja dari sistem (Seidle et al, 2012). Dengan demikian dapat disimpulkan bahwa *Use Case Diagram* menggambarkan fungsionalitas yang diharapkan oleh sistem dengan sifat yang perlu ditekankan adalah "apa" yang akan diperbuat oleh sistem, dan bukan "bagaimana". *Use Case Diagram* merupakan konsep fundamental untuk metode pengembangan berbasis objek. Diagram ini di aplikasikan selama tahap proses analisis dan desain sistem.

Komponen Pembentuk *Use Case Diagram*:

a. Aktor

   Sebuah *use case* tidak dapat berjalan sendiri melakukan suatu tindakan. Aktor dibutuhkan dalam *Use Case Diagram* sebagai pelaku tindakan yang dilakukan oleh *use case*. Dalam *Use Case Diagram* aktor selalu berinteraksi dengan sistem dalam konteks *use case* yang memiliki hubungan dengannya. Aktor tidak hanya dapat berinterkasi dengan satu *use case* saja. Tetapi aktor diperbolehkan untuk berinterkasi dengan banyak *use case* dan suatu *use case* diperbolehkan untuk berinteraksi dengan banyak aktor. Aktor akan selalu berada di luar sistem, artinya pengguna atau user tidak akan pernah menjadi bagian dari sistem. Aktor digambarkan dengan *stick figure,* persegi panjang (berisi informasi tambahan «actor»), atau dengan gambaran simbol bebas yang bisa kita lihat pada Gambar 2.3. Simbol yang digunakan dalam penggambaran aktor dalam *use case* yang spesifik bergantung pada siapa yang membuat model atau yang menggunakan *tool* UML.



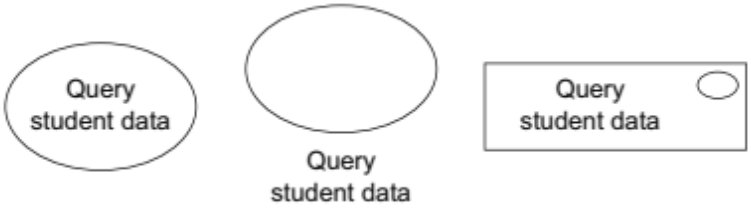
Gambar 2.3 Notasi alternatives Aktor

Sumber: (Seidl et al, 2012)

Beberapa kemungkinan aktor dapat terkait dengan sistem antara lain: memiliki kepentingan dalam sistem dimana terdapat arus informasi yang dibutuhkan, baik yang diterima maupun yang di*input*kan oleh sistem. Orang atau pihak yang akan menerima sistem tersebut. *External resources* yang dibutuhkan oleh sistem. Sistem lain yang berinteraksi dengan sistem yang akan dibangun.

b. *Use Case*

Sebuah *use case* mendeskripsikan fungsi yang diharapkan dari sistem untuk dikembangkan. Ini mencakup sejumlah fungsi yang akan dijalankan ketika menggunakan sistem tersebut. *Use case* memberikan manfaat yang nyata untuk satu atau lebih aktor yang berhubungan dengan *use case* tersebut. *Use case diagram* tidak mencakup struktur internal dan implementasi aktual dari suatu *use case. Use case* merupakan sebuah tindakan tertentu seperti, *login* ke sistem, membuat sebuah daftar belanja, dan sebagainya. Sebuah *use case* biasanya digambarkan dengan bentuk elips. Nama dari *use case* biasanya ditentukan didalam atau dibawah gambar elips. Secara alternatif, sebuah *use case* dapat digambarkan dengan persegi panjang yang berisi nama dari *use case* ditengah dan terdapat elips kecil di pojok kanan atasnya yang bisa kita lihat dalam Gambar 2.4. Namun, bentuk *use case* dengan gambar elips dengan nama didalamnya adalah bentuk yang paling banyak digunakan.



Gambar 2.4 Bentuk alternatif dari use case

Sumber: (Seidl et al, 2012)

c. *Association*

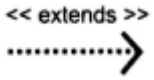
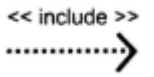
Sebuah aktor dapat berhubungan dengan *use case* dengan menggunakan *associations* untuk mengkespresikan bahwa aktor berkomunikasi dengan sistem dan menggunakan fungsi yang ada. Setiap aktor harus terhubung setidaknya dengan satu *use case* dan setiap *use case* harus terhubung dengan setidaknya satu aktor. Sebuah *association* selalu biner, yang berarti bahwa *association* selalu ditentukan untuk menghubungan satu *use case* dan satu aktor. A*ssociation* digambarkan dengan bentuk garis yang solid. Didalam *use case diagram, associations* terdapat beberapa macam jenis, yaitu:

1. *Generalization, y*aitu sebuah pewarisan yang dapat juga disebut dengan *inheritance*, sebuah elemen dapat memiliki sebuah spesialisasi dari elemen lainnya.
2. *Dependency,* yaitu sebuah elemen yang bergantung dalam beberapa cara ke elemen lainnya.
3. Agregation, yaitu suatu bentuk *association* yang berupa sebuah elemen berisi elemen lainnya.

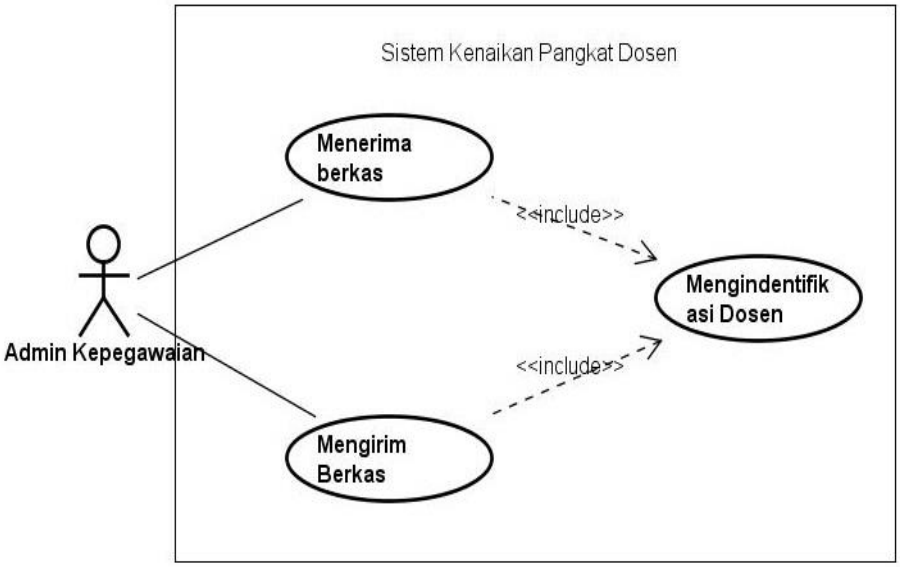
Pada Tabel 2.6 menjelaskan berbagai simbol penting yang akan digunakan dalam penggunaan *use case diagram* nantinya. Terdapat tiga simbol utama dari *use case* yaitu *use case,* aktor, dan relasi dari *use case* yang relasinya dibedakan lagi berdasarkan fungsinya.

Tabel 2.6 Simbol-simbol *Use Case Diagram*

No	Nama	Gambar	Keterangan
1	*Use Case*		Fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor
2.	*Actor*		Orang, proses, atau sistem lain yang dapat berinteraksi dengan sistem yang akan dibuat. Walaupun simbol dari aktor adalah gambar orang akan tetapi belum tentu merupakan orang. Biasanya dinyatakan menggunakan kata benda di awal frase nama aktor.
3	*Association*		Komunikasi antara aktor dan *use case* yang

No	Nama	Gambar	Keterangan
			berpartisipasi pada *use case* atau *use case* memiliki interaksi dengan aktor.
4	*Extend*		Relasi *use case* tambahan ke sebuah *use case* dimana *use case* yang ditambahkan dapat berdiri sendiri walau tanpa *use case* tambahan itu. Mirip dengan prinsip inheritance pada pemograman berorientasi objek, biasanya *use case* tambahan memiliki nama depan yang sama dengan *use case* yang ditambahkan.
5	*Generalization*		Hubungan generalisasi dan spesialisasi (umum-khusus) antara dua buah *use case* dimana fungsi yang satu adalah fungsi yang lebih umu dari lainnya.
6	*Include*		Relasi *use case* tambahan ke sebuah *use case* dimana *use case* yang ditambhakan memerlukan *use case* ini untuk menjalankan fungsi atau sebagai syarat dijalankan *use case* ini

Sumber: (A.S & Shalahuddin, 2011)



Gambar 2.5 Contoh *use case diagram*

Dalam Gambar 2.5 menggambarkan contoh dari penggunaan *Use Case Diagram*. Dari Gambar tersebut teridentifikasi bahwa aktor pada *use case diagram* memiliki peran sebagai seorang admin. Ada beberapa fungsionalitas yang dapat dilakukan pada sistem tersebut, yaitu menerima berkas, mengirim berkas, dan mengidentifikasi data dosen. Akan tetapi admin tidak dapat melakukan tindakan dari *use case* menerima berkas dan mengirim berkas sebelum melakukan tindakan dari *use case* mengindentifikasi data dosen terlebih dahulu.

2.2.5.2 Sequence diagram

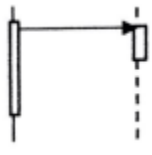
Sequence diagram (diagram urutan) yaitu diagram yang memperlihatkan atau menampilkan hubungan interaksi antar objek didaam sistem yang disusun berdasarkan urutan atau rangkaian waktu. Interaksi antar objek tersebut meliputi aktor, display, dan berupa pesan/*message* (A.S & Shalahuddin, 2011). *Sequence Diagram* menggambarkan interaksi antara objek untuk memenui tugas tertentu. *Sequence Diagram* berfokus pada urutan kronologis dari pesan yang dipertukarkan antara pasangan interaksi (Seidl et al, 2012).

Sequence diagram digunakan dalam tahap analisis dan desain. *Sequence diagram* sering digunakan untuk menggambarkan aliran acara kronologis-terstruktur melalui *use case.* Menurut Booch (2007), s*equence diagram* digunakan untuk melacak pelaksanaan skenario dalam konteks yang sama sebagai diagram komnunikasi (*ommunication diagram). Sequence diagram* menggambarkan perilaku sistem berdasarkan interaksi yang dilakukan antara satu set objek dalam hal pesan yang

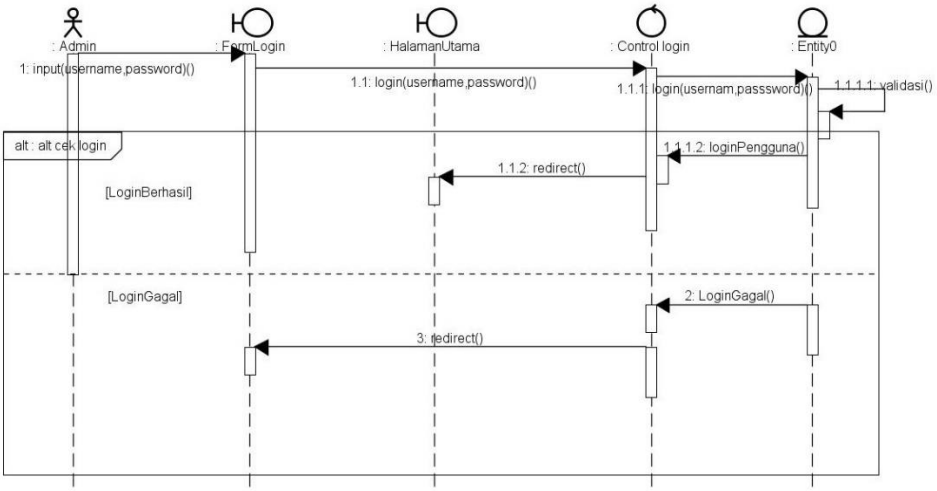
bertukar diantara mereka untuk menghasilkan hasil yang diinginkan. *Sequence diagram* menggaris bawahi bahwa urutan dalam berkirim pesan berdasarkan waktu yang dikerjakan.

Sequence diagram memiliki dua dimensi, yaitu dimensi vertikal yang mewakili waktu serta dimensi horisontal yang mewakili objek yang berbeda. Inisiasi urutan dimulai disudut kiri atas, dan waktu dihalam bawah (dari atas ke bawah). Garis vertikal disebut garis hidup objek *(lifeline).* Pada Tabel 2.7 merupakan simbol-simbol yang terdapat pada s*equence diagram.* Dalam s*equence diagram* terdapat 3 kelas penting dimana nantinya kelas tersebut akan merepresentasikan pada program yang sebenarnya, yaitu *boundary, control,* dan *entity*.

Tabel 2.7 Simbol-simbol *sequence diagram*

No	Nama	Gambar	Keterangan
1	*Actor*		Mengambarkan orang yang berinteraksi dengan sistem
2.	*Boundary*		Menggambarkan antarmuka dan interaksi satu atau lebih aktor dengan sistem
3	*Control*		Menggambarkan perilaku mengatur, mengkoordinasi perilaku sistem dan dinamika dari suatu sistem, menangani tugas utama dan mengontrol alur kerja suatu sistem
4	*Entity*		Menggabmarkan informasi atau data yang harus disimpan oleh sistem
5	*Lifeline*		Mengindikasikan keberadaan sebuah obyek dalam basis waktu
6	*Message*		Menggambarkan pesan / hubungan antar objek, yang menunjukkan urutan kejadian yang terjadi

Sumber: (A.S & Shalahuddin, 2011)



Gambar 2.6 Contoh *Sequence Diagram*

Gambar 2.6 merupakan contoh *sequence diagram* melakukan proses *login* dengan penjelasan pada proses tersebut aktor melakukan *login* dengan meng*input*kan data halaman *login*, halaman *login* pada gambar tersebut menggunkan *Boundary* lalu data akan diteruskan pada proses control setelah itu sistem akan mengecek pada basis data, basis data pada gambar tersebut digambarkan dengan *entity*, setelah melakukan autentifikasi jika data tersebut valid maka sistem akan menampilkan halaman utama, jika tidak sistem akan menampilkan halaman *login* dan kondisi tidak berubah.

2.2.5.3 Class Diagram

Class Diagram digunakan untuk menunjukkan keberadaan kelas yang terdapat pada sistem dan hubungan mereka dalam padangan logis dari sistem (Booch et al, 2007). Konsep *class diagram* digunakan untuk menentukan struktur data dan struktur objek dari suatu sistem (seidl et al, 2015). *Class diagram* terutama didasarkan pada konsep kelas, generalisasi, dan asosiasi. Pada konsep pemograman berorientasi objek, *class diagram* menggambarkan kelas yang terdapat dalam sistem perangkat lunak yang terdiri dari hubungan antar kelas. Satu *class diagram* merupakan representasi dari struktur kelas dari suatu sistem.

Class Diagram digunakan ditahap analisis dan desain. Selama tahap analisis *class diagram* digunakan untuk menunjukkan peran umum dan tanggung jawab entitas yang menyediakan pelaku sistem. Selama tahap desain, *class diagram* digunakan untuk menngambarkan struktur kelas yang akan membentuk arsitektur sistem (Booch et al, 2007).

Class diagram menggambarkan struktur dari sistem, dengan mendefinisikan kelas-kelas yang akan dibuat dalam pembanguna sistem. Kelas memiliki atribut dan operasi atau metode (A.S & Shalahuddin, 2011).

a. Atribut merupakan sekumpkan variabel yang dimiliki oleh suatu kelas.
b. Operasi atau metode merupakan fungsi yang dimiliki oleh suatu kelas.

Suatu atribut dan metode yang ada didalam *class* mempunyai salah satu sifat, berikut ini merupakan sekumpulan sifat yang dapat dimiliki:

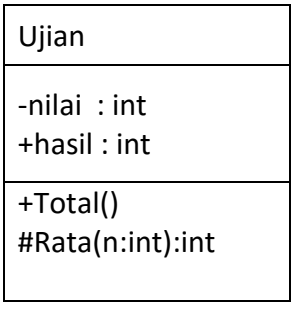
a. *Private* (-)*,* suatu atribut atau metode yang tidak dapat digunakan diluar *class* yang bersangkutan.
b. *Protected* (#), suatu atribut atau metode yang hanya digunakan oleh *class* yang bersangkutan dan anak *class* mewarisinya.
c. *Public* (-), suatu atribut atau metode yang dapat digunakan oleh siapa saja.

Contoh penggunaan atribut pada class diagram:

```
1  Public class ujian {
2            Private nilai;
3            Public int hasil;
4            Public void total(){}
5            Protected int rata(int n){}
6  }
```

Kode 2.1 Contoh kode penggunaan attribut

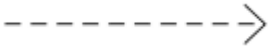
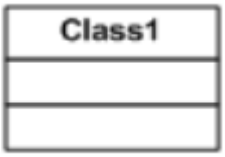
Contoh kode program pada Kode 2.1 jika dibuatkan bentuk *class diagram*-nya akan seperti dalam Gambar 2.7



Gambar 2.7 Contoh Class Diagram

Pada Tabel 2.8 merupakan simbol-simbol hubungan yang terdapat pada class *diagram.* Pada *Class Diagram* Terdapat banyak macam relasi yang dibedakan berdasarkan fungsinya.

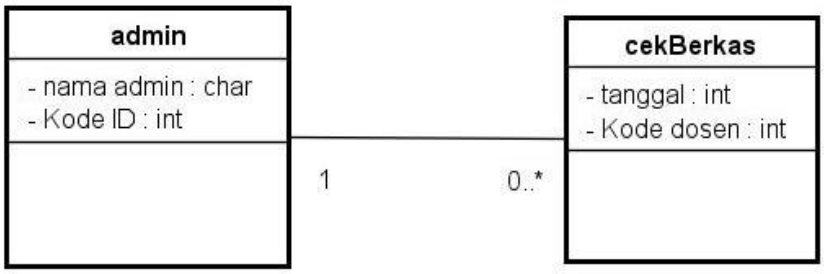
Tabel 2.8 SImbol-simbol *class diagram*

No	Nama	Gambar	Keterangan
1	Asosiasi		Relasi antar kelas dengan makna kedua kelas saling berkaitan
2.	Asosiasi berarah / *Directed Association*		Relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disetai dengan multiplicity
3	Generalisasi		Relasi antar kelas dengan makna generalisasi spesialisasi (umum khusus)
4	Kebergantungan / *Dependency*		Relasi antar kelas dengan makna kebergantungan antar kelas
5	*Agregation*		Relasi antar kelas dengan makna semua bagian (whole-part)
6	*Class*		Menggambarkan seseuatu yang mengkapsulkan informasi di *class* menampung nama *class,* atribut dan *method*

Sumber: (A.S & Shalahuddin, 2011)

Dalam Gambar 2.8 menunjukkan hubungan antar kelas admin dengan kelas cekBerkas lengkap dengan atribut yang dimiliki oleh masing-masing kelas. Atribut dari admin adalah nama dan kode, sedangkan atribut dari cekBerkas adalah tanggal dan kode dosen, angka 1 dan 0 menjelaskan *multiplicity,* yang memiliki pengertian 1 aktor admin dapat mengerjakan 0 atau banyak dari cekBerkas. Contoh lain notasi *multiplicity* yang memberikan gambaran sejumlah instan yang akan ditampung dalam *class* adalah:

- 0...1 = nol atau satu instan
- 0...* = nol atau banyak instan
- * = memiliki banyak instan
- 1 = hanya ada satu instan
- 1...* = satu atau banyak instan



Gambar 2.8 Contoh Class Diagram

2.2.6 Pengujian Perangkat Lunak

Pengujian perangkat lunak juga dikenal sebagai tahapan verikasi dan validasi yang merupakan proses untuk memeriksa bahwa perangkat lunak yang telah dibangun memenuhi requirement dan spesifikasi yang telah ditentukan sebelumnya (Bassil, 2012). Verikasi adalah proses evaluasi *software* untuk menetukan apakah produk dari tahap pengembangan yang diberikan memenuhi kondisi yang ditentukan pada tahap awal pengembangan. Sementara itu validasi adalah proses evaluasi software selama proses atau pada akhir proses pembangunan untuk menentukan apakah software yang dikembangkan memenuhi *requirement* yang telah ditentukan. Selain itu, tahap pengujian adalah tahapan untuk melakukan *debugging* dimana *bug* dan gangguan pada sistem ditemukan, dikoreksi dan disempurnakan sesuai *requirement*.

Menurut Sommerville (2011), setiap unit dari setiap program yang sudah ada harus diintergrasikan dan diuji sebagai satu kesatuan yang utuh untuk memastikan apakah kebutuhan sistem sudah terpenuhi. Tahap pengujian sistem memiliki dua tujuan yang berbeda, seperti (Sommerville, 2011):

1. Untuk menunjukkan kepada *developer*/pengembang dan customer bahwa sistem yang dikembangkan telah memenuhi *requirement.*
2. Untuk menemukan kesalahan dalam sistem atau tidak sesuai dengan spesifikasi sistem yang sudah ditentukan.

Teknik atau pengujian sistem yang akan digunakan peneliti dalam menguji kebutuhan fungsional yaitu menggunakan *black-box* testing.

2.2.6.1 *Black Box Testing*

Black box testing berfokus pada kebutuhan fungsional sistem. Metode pengujian *black box* mengijinkan pengembang untuk menghasilkan sekumpulan kondisi masukan pada sistem yang sepenuhnya akan melaksanakan kebutuhan

fungsional untuk sebuah program (Pressman, 2010). Pengujian *black box* merupakan suatu pendekatan komplementer seperti mengungkap perbedaan perbedaan *error* yang terjadi pada suatu *class.* Berikut ini merupakan kesalahan-kesalahan yang berusaha ditemukan dalam pengujian *black box*: Kesalahan pada fungsi aatau terjadi fungsi yang hilang. Kesalahan pada desain antarmuka. Kesalahan di dalam struktur data atau dalam akses ke basis data eksternal. Kesalahan dalam perilaku atau kesalahan kinerja. Inisialisasi atau pemusatan kesalahan

Pengujian dengan metode *black box* tidak dituntun untuk mempunyai pemahaman terhadap struktur internal atau kode dari suatu sistem dan hanya berfokus kepada keluaran yang berasal dari kebutuhan yang telah ditentukan (Irena, 2008). Penguji dalam *black box testing* hanya mengetahui *input* yang dapat diberikan kepada sistem dan apa *output* yang harus diberikan oleh sistem. Dengan kata lain, dasar untuk menentukan *test case* dalam *black box testing* adalah *requirement* dan spesifikasi dari sistem yang telah ditentukan (Agarwal, 2010).

2.2.6.2 Compatibility testing

Compatibility testing merupakan metode pengujian yang berfokus pada kebutuhan non-fungsional. Pengembang menggunakan *compability testing* untuk memastikan apakah sistem tersebut berjalan dengan baik di berbagai lingkungan yang berbeda (Yoon, 2008). Beberapa tipe pengujian pada *compatibility testing* antara lain (Guru99, 2016) : *hardware*, *operating system*, *software*, *network*, *browser*, *devices*, *mobile*, *version*

Tipe pengujian *compatibility* yang akan digunakan pada penelitian ini adalah tipe pengujian pada sisi *browser*. Pengujian ini dilakukan diberbagai lingkungan *browser* yang berbeda antara lain *firefox, google chrome, internet explorer, Microsoft edge, safari, browser android, browser ios.* Beberapa *tools* yang dapat digunakan untuk mendukung pengujian ini antara lain *sortsite, secure platform, virtual desktop, selenium, imacros, sahi, ranorex studion,* dan sebagainya.

Tools yang akan digunakan pada penelitian ini adalah *sortsite. Sortsite* dipilih karena mampu dalam menguji perangkat lunak berbasis web secara menyeluruh pada setiap halaman yang dimiliki. Selain itu, *Sortsite* mampu menguji perangkat lunak berbasis web pada beberapa *checkpoint* diantaranya *accessibility, broken link, compatibility, search engine optimation, privacy, web standart,* dan *usability* (PowerMapper, 2016).