BAB II
LANDASAN TEORI

2.1 Software Engineering

Software engineering adalah disiplin teknik yang berkaitan dengan semua aspek produksi *software*. *Software engineering* pertama kali diusulkan pada tahun 1968 pada konferensi yang diadakan untuk membahas apa yang kemudian disebut *software crisis* (Naur and Randell, 1968). Sepanjang tahun 1970-an dan 1980-an, berbagai Teknik dan metode rekayasa perangkat lunak baru dikembangkan, seperti *structured programming, information hiding,* dan *object-oriented development* (Sommerville, 2010).

Software development life cycle (SDLC) adalah fase yang memberikan pemahaman umum tentang proses pembuatan *software*. Sebagaimana *software* akan direalisasikan dan dikembangkan dari pemahaman bisnis dan fase elisitasi persyaratan untuk mengkonversi ide-ide bisnis dan persyaratan ke dalam fungsi dan fitur sampai mencapai kebutuhan bisnis. *Software engineer* yang baik harus memiliki pengetahuan yang cukup tentang cara memilih model SDLC berdasarkan konteks proyek dan persyaratan bisnis (Sami, 2012).

Menurut Zave (1997), *requirements engineering* adalah cabang dari *software engineering* yang berkaitan dengan tujuan dunia nyata untuk fungsi dan Batasan pada sistem *software*. Hal ini juga berkaitan dengan hubungan faktor-faktor ini dengan spesifikasi perikalu *software* yang tepat dan evolusi mereka dari waktu ke waktu. Tahap ini digunakan untuk menerjemahkan kebutuhan dan keinginan pengguna sehingga *software* dalam spesifikasi yang lengkap.

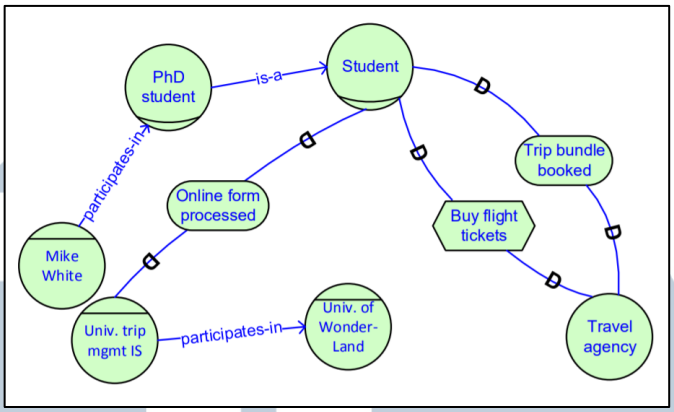
2.2 Requirements Engineering

Requirements engineering adalah proses penentuan ekspektasi pengguna untuk produk baru atau yang dimodifikasi (Margaret Rouse, 2016). Tahap ini digunakan untuk menerjemahkan kebutuhan dan keinginan pengguna sehingga *software* dalam spesifikasi yang lengkap. Oleh karena itu, pentingnya *requirements engineering* sangat besar untuk mengembangkan *software* yang efektif dan mengurangi kesalahan *software* dalam tahap awal pengembangan (Chakraborty et al., 2012).

Framework yang populer saat ini yaitu *framework* iStar 2.0 yang tidak didefinisikan sebagai bahasa baru, tetapi penyempurnaan konsep *framework* iStar (López dkk, 2016). Framework iStar adalah *framework* dalam *software engineering* yang mendukung permodelan sistem dan organisasi *socio-technical* sistem dan organisasi (Franch dkk, n.d.). iStar berfokus pada yang dimaksud (*why?*), sosial (*who?*), dan strategis (*how? How else?*). i* telah diterapkan di banyak area misalnya, *healthcare*, *security analysis*, *eCommerce* (Dalpiaz dkk, 2016).

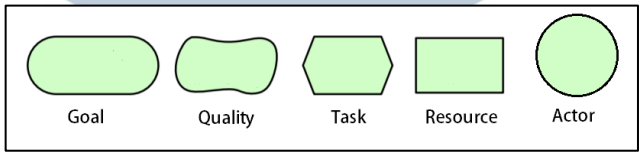
2.3 Strategic Dependency Model

Strategic Dependency *model* terdiri dari serangkaian node dan tautan. Setiap node mewakili actor dan setiap tautan antara dua actor menunjukan bahwa satu actor berganti pada yang lain untuk sesuatu agar yang pertama dapat mencapai beberapa tujuan (Yu dan Liu, 2001).



Gambar 2.1 Strategic Dependency Model dari *Travel Reimbursement* (Dalpiaz dkk, 2016)

Pada Gambar 2.1 Menunjukan contoh dari skenario pengembalian dana *travel* menggunakan Strategic Dependency *model* pada diagram iStar 2.0. Terdapat lima jenis bentuk pada diagram iStar 2.0 yang menggunakan Strategic Dependency *model* yaitu seperti pada gambar 2.2.



Gambar 2.2 Bentuk dari Strategic Dependency Model (Dalpiaz dkk, 2016)

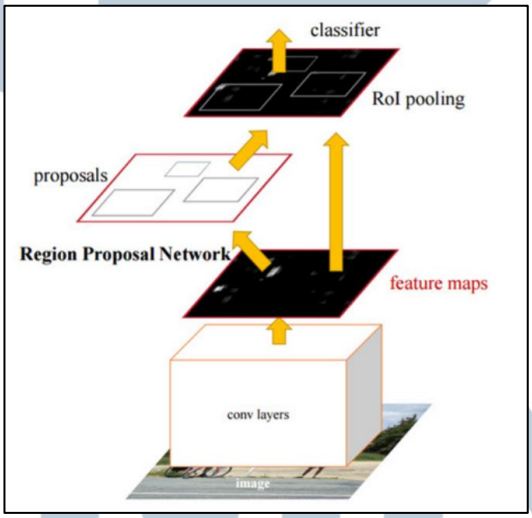
Pada Gambar 2.2 *Goal* adalah keadaan yang ingin dicapai olek *actor* dan memiliki kriteria pencapaian yang jelas. *Quality* adalah atribut yang diinginkan *actor* pada tingkap pencapaian tertentu. *Task* mewakili tindakan yang ingin dieksekusi oleh *actor*, biasanya dengan mencapai beberapa tujuan. *Resource* adalah entitas fisik atau informasi yang dibutuhkan *actor* untuk melakukan tugas. *Actor* adalah entitas aktif dan otonom yang bertujuan mencapai tujuan mereka dengan menggunakan pengetahuan mereka dan bekerja sama dengan *actor* lain. *Generic actor* merupakan *actor* yang tidak memiliki spesialisasi.

2.4 Object Detection

Object detection bertujuan untuk mendeteksi semua objek dari *class* yang diketahui dalam gambar. *Object detection* membangun sebuah model dari sebuah *object class* dari contoh *training*. Biasanya dalam satu gambar memiliki objek yang sedikit tetapi sangat besar kemungkinan lokasi yang ada. Setiap deteksi memberikan beberapa informasi seperti lokasi dari objek, sebuah lokasi dan ukuran, dan mengkotakannya objek yang ditemukan (Amit dan Felzenszwalb, n.d.).

2.5 Faster Region-based Convolution Neural Network

Faster R-CNN merupakan pengembangan dari R-CNN. Sebagai generasi terbaru dari metode *object detection* menunjukkan hasil yang mengesankan pada *object detection*.

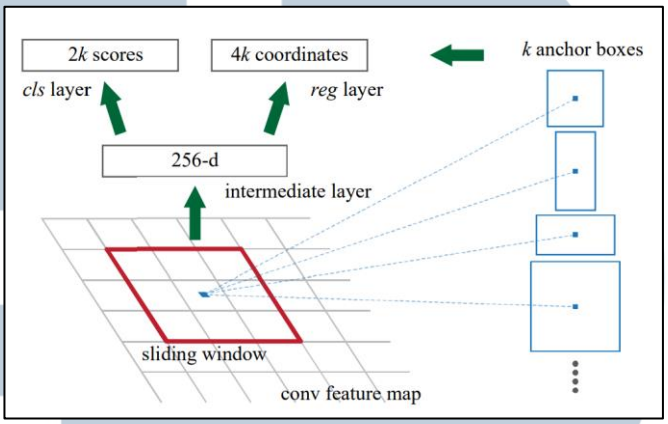


Gambar 2.3 Arsitektur dari Faster R-CNN (Abbas, 2018)

Gambar 2.3 menunjukkan arsitektur dari Faster R-CNN. Faster R-CNN memiliki langkah-langkah sebagai berikut.

a) *Region Proposal Network* (RPN) yaitu tugas cepat yang berguna untuk mencari pada *input* gambar kemungkinan lokasi dari objek. Posisi dari objek yang ada pada gambar memiliki kemungkinan objek dibatasi dari

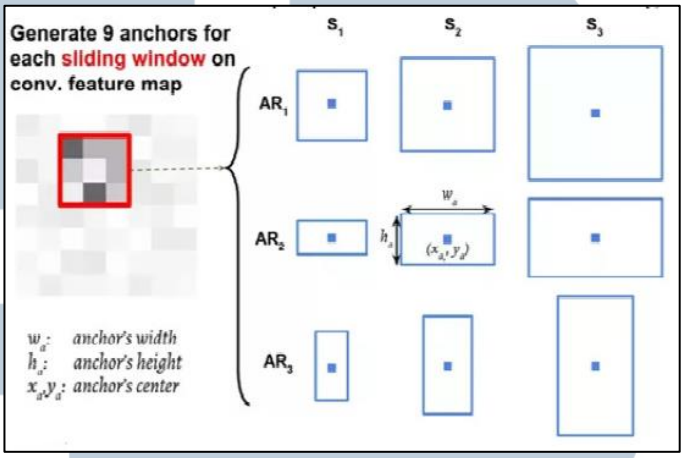
wilayah yang diketahui sebagai *region of interest* (ROI). RPN mengambil gambar dari berbagai ukuran sebagai input dan output sekumpulan proposal objek persegi panjang, masing-masing dengan skor objektivitas.



Gambar 2.4 Region Proposal Network (RPN) (Abbas, 2018)

Pada Region Proposal Network awalnya gambar dimasukkan ke dalam jaringan Convolutional Neural Network. Gambar input diteruskan ke jaringan *convolutional layer* terakhir yang menampilkan *feature map*. *Sliding window* ditempatkan pada setiap bagian dari *feature map*. *Sliding window mask* biasanya diambil dari ukuran *mask* n*n. Sesuai dengan setiap *sliding window* maka *anchor* dibuat (Abbas, 2018). Di setiap lokasi *sliding window*, secara bersamaan memprediksi beberapa *region proposal*, di mana jumlah proposal maksimum yang mungkin untuk setiap lokasi dilambangkan sebagai k. Layer *reg* memiliki output 4k yang mengkodekan koordinat kotak k, dan layer *cls* menghasilkan skor 2k yang memperkirakan probabilitas objek atau tidak objek untuk setiap proposal. Setiap *anchor* diposisikan di tengah dari *sliding windows*. Secara standar menggunakan 3 skala dan 3 rasio aspek yang menghasilkan k = 9 *anchor* pada setiap *sliding windows* (Ren dkk, 2016). Klasifikasi menunjukkan

probabilitas 0 atau 1 yang menunjukkan apakah wilayah tersebut berisi objek atau tidak dengan p^* = 1 jika IoU > 0,7, p^* = -1 jika IoU < 0,3, p^* = 0 jika selain itu (Abbas, 2018).



Gambar 2.5 Tiga Anchor dengan Aspect ratio dan Scaling (Abbas, 2018)

b) *Classification* yaitu mengklasifikasikan *regions of interest* yang telah diidentifikasi dari langkah di atas menjadi koresponden dari banyak *class*.

2.6 Color-to-grayscale

Color-to-grayscale adalah perubahan citra RGB menjadi citra *grayscale*. *Grayscale* adalah suatu citra dimana nilai dari setiap *pixel* merupakan sample tunggal. Citra yang ditampilkan dari citra jenis ini terdiri atas warna abu-abu, bervariasi pada warna hitam pada bagian yang intensitas terlemah dan warna putih pada intensitas terkuat (Fatta, 2007). Terpadat beberapa algoritma untuk melakukan *color-to-grayscale* salah satunya adalah Gleam. Rumus dari color-to-grayscale Gleam sebagai berikut (Kanan dan Cottrell, 2012).

$$Gleam = \frac{1}{3}(R' + G' + B') \quad \ldots (2.1)$$

Pada penelitian Kanan dan Cottrell (2012) mengatakan bahwa untuk pengenalan objek dan pengenalan wajah, Gleam hampir selalu berkinerja paling baik.

2.7 Salt and Pepper Noise

Salt and Pepper noise dapat merusak gambar di mana piksel rusak mengambil tingkat abu-abu maksimum atau minimum (Esakkirajan dkk, 2011). Untuk gambar yang rusak oleh Salt and Pepper *noise* (masing-masing, noise bernilai acak), piksel noise hanya dapat mengambil nilai maksimum dan minimum (masing-masing, nilai acak apa pun) dalam rentang dinamis (Chan dkk, 2005).

2.8 Tensorflow Object Detection API

Tensorflow Object Detection API adalah kerangka kerja sumber terbuka yang dibangun di atas TensorFlow yang membuatnya mudah untuk membangun, melatih, dan menggunakan model deteksi objek. Menciptakan model pembelajaran mesin yang akurat yang mampu melokalkan dan mengidentifikasi banyak objek dalam satu gambar tetap menjadi tantangan utama dalam visi komputer. Di Google, kami tentu menemukan basis kode ini berguna untuk kebutuhan visi komputer kami, dan kami berharap Anda juga akan melakukannya (Huang dkk, 2017).

2.9 Confusion Matrix

Confusion matrix adalah sebuah tabel yang menyatakan jumlah data uji yang benar diklasifikasikan dan jumlah data uji yang salah diklasifikasikan. Contoh confusion matrix untuk klasifikasi biner ditunjukan pada Tabel 2.1.

Tabel 2.1 Confusion Matrix untuk klasifikasi biner (Indriani, 2014)

		Kelas Prediksi	
		1	0
Kelas Sebenarnya	1	TP	FN
	0	FP	TN

True Possitive (TP) yaitu jumlah dokumen dari kelas 1 yang benar dan diklasifikasikan sebagai kelas 1. *True Negative* (TN) yaitu jumlah dokumen dari kelas 0 yang benar diklasifikasikan sebagai kelas 0. *False Positive* (FP) yaitu jumlah dokumen dari kelas 0 yang salah diklasifikasikan sebagai kelas 1. *False Negative* (FN) yaitu jumlah dokumen dari kelas 1 yang salah diklasifikasikan sebagai kelas 0 (Indriani, 2014). Berdasarkan nilai *True Possitive* (TP), *True Negative* (TN), *False Positive* (FP), *False Negative* (FN) dapat diperoleh nilai akurasi, presisi, *recall*.

Nilai akurasi menggambarkan seberapa akurat sistem dapat mengklasifikasikan data secara benar. Nilai presisi menggambarkan jumlah data kategori positif yang diklasifikasikan secara benar dibagi dengan total data yang diklasifikasikan positif. Sementara itu, *recall* menunjukkan berapa persen data kategori positif yang terklasifikasikan dengan benar oleh sistem. Rumus dari akurasi, presisi, *recall* adalah sebagai berikut (Solichin, 2017).

$$Akurasi = \frac{TP+TN}{TP+TN+FP+FN} * 100\% \quad \dots (2.2)$$

$$Presisi = \frac{TP}{FP+TP} * 100\% \quad \dots (2.3)$$

$$Recall = \frac{TP}{FN+TP} * 100\% \quad \dots (2.4)$$

F1-score diperlukan saat ingin mencari keseimbangan antara *precision* dan *recall*. F1-score mungkin menjadi ukuran yang lebih baik untuk digunakan jika kita perlu mencari keseimbangan antara *precision* dan *recall*. Rumus F1-*score* sebagai berikut (Shung, 2018).

$$F1 = 2 * \frac{Precision*Recall}{Precision+Recall} \quad \dots (2.5)$$