BINUS UNIVERSITY ONLINE LEARNING

People
Innovation
Excellence

LECTURE NOTES

Software Engineering

Minggu 5

Software Quality Assurance

BINUS UNIVERSITY ONLINE LEARNING

People
Innovation
Excellence

LEARNING OUTCOMES

Setelah menyelesaikan pembelajaran ini, mahasiswa akan mampu:

- ☑ LO3 – Mendemonstrasikan penjaminan mutu suatu *software*

Outline Materi (Sub-Topic):

1. *Software Quality*
2. *The Software Quality Dilemma*
3. *Achieving Software Quality*
4. *Review Techniques*
5. *Defect Amplification*
6. *Review Metrics*
7. *Informal Reviews*
8. *Formal Technical Reviews*
9. *Software Reliability*
10. Studi kasus

ISI MATERI

1. *Software Quality*

Apakah yang dimaksud dengan kualitas? Kualitas berarti seberapa jauh suatu produk dapat memenuhi kebutuhan yang ditentukan di awal pengerjaan. Produk yang tidak berkualitas akan menimbulkan masalah, di antaranya:

- Diperlukannya *rework* untuk produk tersebut
- Pertambahan biaya
- Pengaruh kepada reputasi produk dan perusahaan
- Berdampak pada kepuasan pelanggan

Bahkan ada dampak yang lebih berat lagi jika terjadi masalah yang berhubungan dengan hukum, atau pun berkaitan dengan nyawa manusia. Misalnya, suatu produk perangkat lunak yang terdapat pada otomotif, atau kereta api. Jika terjadi kesalahan algoritma yang menyebabkan terjadinya kecelakaan, tentunya ini sangat berbahaya.

Membangun perangkat lunak yang berkualitas tentunya menjadi hal yang sangat penting untuk dilakukan. Terdapat dua kategori kegiatan yaitu: *software quality assurance (SQA)* dan *software quality control (SQC).* SQA berhubungan dengan memastikan bahwa kegiatan yang berhubungan dengan kualitas perangkat lunak dikerjakan sesuai dengan rencana, sedangkan SQC sendiri merupakan bentuk kegiatan untuk melakuan testing terhadap perangkat lunak yang dibangun.

Sebelum melakukan kegiatan pelaksanaan penjaminan mutu perangkat lunak, diperlukan perencanaan terhadap aktivitas yang perlu dilakukan, di antara nya:

- Kegiatan apa yang perlu dilakukan terhadap kegiatan yang berkenaan terhadap kualitas

- Metode apa yang dipakai untuk melakukan verifikasi, validasi dan testing
- Apa saja metrics yang dipakai di dalam mengukur suatu kualitas perangkat lunak, misalnya:
    - Performance
    - Availability
    - Security
    - Usability
    - Reliability
- Bagaimana membandingkan antara hasil yang didapatkan dengan metrics yang direncanakan di awal
- Alat bantu apa yang cocok digunakan

2. *The Software Quality Dilemma*

Di dalam menentukan suatu derajat atau *metrics* kualitas dari perangkat lunak, juga harus dipertimbangkan dengan penjadwalan dan biaya yang berhubungan dengan pencapaian kualitas tersebut.

Jika kita mengembangkan sebuah *software* yang memiliki kualitas buruk, maka hal tersebut akan menjadi sebuah kegagalan dimana tidak akan ada yang mau membeli *software* tersebut. Di sisi lain, jika kita menghabiskan waktu terlalu lama dengan usaha dan biaya yang sangat besar untuk mengembangkan sebuah *software* yang sempurna, *software* yang dihasilkan akan terlalu mahal sehingga kita tidak dapat memasarkannya. Ini adalah salah satu dilemma terbesar dalam pengembangan *software*. Sehingga dalam *software* engineering, yang menjadi target adalah software yang “cukup baik” sehingga tidak akan

ditolak oleh pasar dengan menggunakan sumber daya yang efisien dalam pengembangannya (biaya, waktu, sumber daya manusia, dll). *Software* yang "cukup baik" memberikan fungsi dan *feature* dengan kualitas tinggi yang diinginkan oleh *user*, namun juga memberikan fungsi-fungsi spesifik yang kurang jelas yang masih memiliki *bugs*.

3. *Achieving Software Quality*

Kualitas perangkat lunak didefinisikan sebagai konformansi terhadap kebutuhan fungsional dan kinerja yang dinyatakan secara eksplisit, standar perkembangan yang didokumentasikan secara eksplisit, dan karakteristik implisit yang diharapkan bagi semua perangkat lunak dikembangkan secara profesional. definisi tersebut berfungsi untuk menekankan tiga hal penting, yaitu:

1. Kebutuhan perangkat lunak merupakan fondasi yang melaluinya kualitas diukur.
2. Standar yangtelah ditentukan menetapkan serangkaian kriteria pengembangan yang menuntun cara perangkat lunak direkayasa.
3. Ada serangkaian kebutuhan implisit yang sering dicantumkan (misalnya kebutuhan akan kemampuan pemeliharaan yang baik).
4. Kelompok SQA berfungsi sebagai perwakilan inhouse pelanggan, yaitu orang yang akan melakukan SQA harus memperhatikan perangkat lunak dari sudut pandang pelanggan.

Kelompok SQA harus dapat menjawab pertanyaanpertanyaan dibawah ini untuk memastikan bahwa kualitas perangkat lunak benar-benar terjaga.

- Apakah perangkat lunak cukup memenuhi faktor kualitas
- Sudahkah pengembangan perangkat lunak dilakukan sesuai dengan standar yang telah ditetapkan sebelumnya?
- Sudahkah disiplin teknik dengan tepat memainkan perannya sebagi bagian dari aktivitas SQA?

Aktivitas SQA Jaminan kualitas perangkat lunak terdiri dari berbagai tugas yang berhubungan dengan dua konstituen yang berbeda:

- perekayasa perangkat lunak yang mengerjakan kerja teknis
- kelompok SQA yang bertanggung jawab terhadap perencanaan jaminan kualitas, kesalahan, penyimpanan rekaman, analisis, dan pelaporan.

Tugas kelompok SQA adalah membantu tim rekayasa perangkat lunak dalam pencapaian produk akhir yang berkualitas tinggi.

Aktivitas yang dilakukan (atau difasilitasi) oleh kelompok SQA yang independen: Menyiapkan rencana SQA untuk suatu proyek. Rencana tersebut mengindentifikasikan hal-hal berikut:

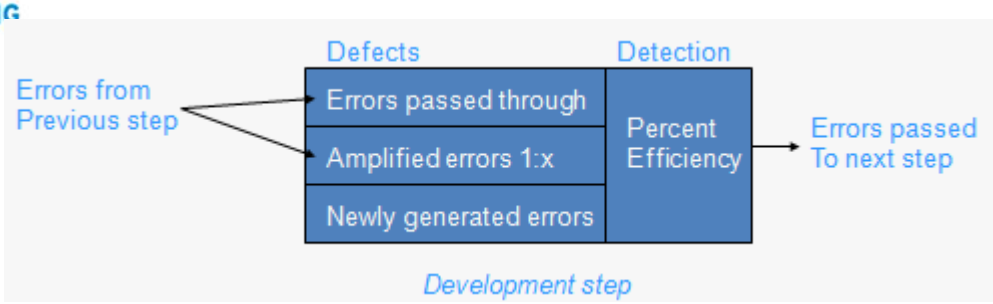
- Evaluasi yang dilakukan
- Audit dan kajian yang dilakukan
- Standar yang dapat diaplikasikan pada proyek
- Prosedur untuk pelaporan & penelusuran kesalahan
- Dokumen yang dihsilkan oleh kelompok SQA
- Jumlah umpan balik yang diberikan pada tim proyek perangkat lunak

4. *Review Techniques*

Kajian perangkat lunak merupakan salah satu aktivitas SQA yang terpenting. Kajian perangkat lunak adalah suatu filter bagi proses rekayasa perangkat lunak, yaitu kajian yg diterapkan pd berbagai titik selama pengembangan PL & berfungsi untuk mencari kesalahan yg kemudian akan dihilangkan. Kajian perangkat lunak berfungsi untuk “memurnikan” produk kerja perangkat lunak yang terjadi sebagai hasil dari analisis, desain, dan pengkodean.

5. *Defect Amplification*

Sebuah *defect amplification* model dapat digunakan untuk mengilustrasikan pembuatan dan pendeteksian error dalam proses perancangan dan programming seperti pada gambar di bawah ini.



6. *Review Metrics*

Berikut ini adalah *metric* yang dapat digunakan untuk melakukan *review* terhadap *software* berdasarkan *effort* dan total jumlah *error* yang ditemukan:

- $E_{review} = E_p + E_a + E_r$
- $Err_{tot} = Err_{minor} + Err_{major}$

Defect *density* menggambarkan jumlah *error* yang ditemukan per unit kerja dari produk yang di*review*:

- *Defect density* = Err_{tot} / WPS

Keterangan:

Ep : usaha persiapan (orang/jam) yang dibutuhkan untuk melakukan *review* dari sebuah produk kerja sebelum rapat *review* yang sebenarnya dilakukan

Ea : usaha penilaian (orang/jam) yang dikeluarkan selama proses *review* yang sebenarnya

Er : usaha (orang/jam) yang didedikasikan untuk melakukan koreksi pada *error* yang ditemukan selama proses *review*

WPS : *work product size* merupakan ukuran dari produk kerja yang di*review* (seperti UML model atau jumlah halaman dari dokumen-dokumen, jumlah baris dari *source code*)

Err_{minor} : jumlah *error* yang ditemukan yang dikategorikan sebagai *error minor*.

Err_{major} : jumlah *error* yang ditemukan yang dikategorikan sebagai *error major*.

7. *Informal Reviews*

Informal review meliputi kegiatan berikut:

- *Desk check* sederhana dari produk kerja *software engineering* yang dilakukan dengan rekan kerja
- *Casual meeting*, meliputi lebih dari 2 orang yang bertujuan untuk melakukan *review* terhadap produk kerja
- Melakukan *review* dari pair programming.

8. *Formal Technical Reviews*

FTR (Formal Technical Review) adalah aktivitas jaminan kualitas perangkat lunak yang dilakukan oleh perekayasa perangkat lunak. Kajian teknik formal atau *walktrough* adalah pertemuan kajian yang disesuaikan dengan kebutuhan yang terbukti sangat efektif untuk menemukan kesalahan.

Keuntungan utama kajian teknis formal adalah penemuan kesalahan sejak awal sehingga tidak berlanjut ke langkah selanjutnya dalam proses perangkat lunak.

Tujuan FTR adalah

1. Menemukan kesalahan dlm fungsi, logika, / implementasinya dlm berbagai representasi PL;
2. Membuktikan bahwa perangkat lunak di bawah kajian memenuhi syarat;

3. Memastikan bahwa PL disajikan sesuai dgn standar yg sudah ditentukan sebelumnya;
4. Mencapai perangkat lunak yg dikembangkan dengan cara yang seragam;
5. Membuat proyek lebih dapat dikelola.

FTR berfungsi sebagai dasar pelatihan yang memungkinkan perekayasa yunior mengamati berbagai pendekatan yang berbeda terhadap analisis perangkat lunak, desain, dan implementasi. FTR juga berfungsi untuk mengembangkan backup dan kontinuitas karena sejumlah orang mengenal baik bagian-bagian perangkat lunak yang tidak mereka ketahui sebelumnya. Masing-masing FTR dilakukan sebagai suatu pertemuan dan akan berhasil hanya bila direncanakan, dikontrol dan dihadirkan dengan tepat. Dalam paragraf berikut, panduan yang mirip dengan walktrough disajikan sebagai kajian teknis formal representatif.

Pertemuan Kajian Tanpa memperhatikan format FTR yang dipilih, setiap pertemuan kajian harus mematuhi batasan-batasan berikut ini:

- Antara 3 & 5 orang (khususnya) harus dilibatkan dalam kajian;
- Persiapan awal harus dilakukan, tetapi waktu yang dibutuhkan harus tidak lebih dari 2 jam dari kerja bagi setiap person
- Durasi pertemuan kajian harus kurang dari 2 jam Pertemuan kajian dihadiri oleh pimpinan kajian, pengkaji, dan prosedur.

Salah satu dari pengkaji berperan sebagai pencatat, yaitu seseorang yang mencatat semua masalah penting yang muncul selama pengkajian. FTR dimulai dengan pengenalan agenda dan pendahuluan dari prosedur. Bila ada masalah kesalahan ditemukan akan dicatat. Pada akhir kajian, semua peserta FTR yang hadir harus memutuskan apakah akan

1. Menerima produk kerja tanpa modifikasi lebih lanjut,
2. Menolak produk kerja sehubungan dengan kesalahan yangada (sekali dbetulkan, kajiann lain harus dilakukan), atau

3. Menerima produk kerja secara sementara (kesalahan minor telah terjadi & harus dikoreksi, tetapi kajian tambahan akan diperlukan). Keputusan kemudian dibuat.

Semua peserta FTR melengkapinya dengan tanda tangan yang menunjukkan partisipasi mereka dalam kajian serta persetujuan mereka terhadap pertemuan tim kajian. Pelaporan Kajian dan Penyimpanan Rekaman Selama FTR, seorang pengkaji (pencatat) secara aktif mencatat semua masalah yang sudah dimunculkan, yang kemudian dirangkum pada akhir pertemuan sehingga dihasilkan daftar masalah kajian. Sebagai tambahan, laporan rangkuman kajian yang sederhana telah diselesaikan di mana rangkuman kajian merupakan jawaban dari tiga pertanyaan berikut:

1. Apa yang dikaji?
2. Siapa yang melakukan?
3. penemuan apa yang dihasilkan dan apa kesimpulannya?

Daftar masalah kajian mempunyai dua tujuan:

1. Mengidentifikasi area masalah pada produk,
2. Daftar item kegiatan yang menjadi petunjuk bagi prosedur saat koreksi dilakukan. Daftar masalah biasanya dilampirkan pada laporan.

Pedoman Kajian Pedoman untuk melakukan kajian teknis formal harus dilakukan sebelumnya, didistribusikan kepada semua pengkaji, disetujui, dan kemudian dilaksanakan. Kajian yang tidak terkontrol sering dapat menjadi lebih buruk daripada bila tidak ada kajian sama sekali. Berikut ini serangkaian pedoman minimum untuk kajian teknis formal:

1. Kajian produk, bukan produser.
2. Menetapkan agenda dan menjaganya.
3. Membatasi perdebatan dan bantahan.

4. Menetapkan area masalah, tetapi tidak tergoda untuk menyelesaikannya setiap masalah yang dicatat.
5. Mengambil catatan tertulis.
6. Membatasi jumlah peserta dan mewajibkan persiapan awal.
7. Mengembangkan daftar bagi masingmasing produk kerja yang akan dikaji.
8. Mengalokasikan sumber-sumber daya dan jadwal waktu untuk FTR.
9. Melakukan pelatihan bagi semua pengkaji.
10. Mengkaji kajian awal Anda.

9. *Software Reliability*

Reliabilitas perangkat lunak, tidak seperti faktor kualitas yang lain, dapat diukur, diarahan, dan diestimasi dengan menggunakan data pengembangan historis. Reliabilitas perangkat lunak didefinisikan dalam bentuk statistik sebagai "kemungkinan operasi program komputer bebas kegagalan di dalam suatu lingkungan tertentu dan waktu tertentu". Kapan saja reliabilitas perangkat lunak dibicarakan, selalu muncul pertanyaan yang sangat penting : Apa yang dimaksudkan dengan bentuk "kegagalan?" dalam konteks dan banyak diskusi mengenai kualitas dan reliabilitas perangkat lunak, kegagalann adalah ketidaksesuaian dengan kebutuhan perangkat lunak. Kegagalan hanya akan mengganggu atau bahkan merupakan bencana. Satu kegagalan dapat diperbaiki dalam beberapa detik sementara kesalahan yang lain mungkin membutuhkan waktu pembetulan berminggu-minggu atau bahkan berbulan-bulan. Pembetulan satu kegagalan kenyataannya dapat menghasilkan kesalahan lain yang baru yang mungkin akan membawa lagi kesalahan yang lain lagi. Pengukuran Reliabilitas dan Availabilitas Kerja awal dalam reliabilitas perangkat lunak berusaha mengekstrapolasi matematika teori reliabitas perangkat keras. Sebagian besar model reliabilitas yang berhubungan dengan perangkat keras didasarkan pada kegagalan sehubungan dengan keusangan (*wear*), bukan kesalahan karena cacat desain. Dalam perangkat keras, kegagalan sehubungan dengan keusangan fisik (misalnya

pengaruh suhu, korosi, kejutan) lebih banyak terjadi daripada kegagalan karena isu. Akan tetapi, yang terjadi pada perangkat lunak adalah hal yang sebaliknya. Kenyataannya, semua kegagalan perangkat lunak dapat ditelusuri ke dalam desain atau masalah implementasi; keusangan tidak tercakup. Masih ada perdebatan yang terjadi di seputar hubunan antara konsep kunci dalam reliabilitas perangkat keras dan kemampuan aplikasinya terhadap perangkat lunak. Meskipun ada hubungan yang tidak dapat dibantah, namun sangat penting untuk memprtimbangkan beberapa konsep sederhana yang berlaku untuk kedua sistem elemen tersebut. Bila kita andaikan suatu sistem yang berbasis komputer, pengukuran reliabilitas secara sederhana adalah berupa *mean time between failure* (MTBF), dimana: MTBF = MTTF + MTTR (Akronim MTTF adalah *mean time to failure* dan MTR berarti *mean time to repair*.) Banyak peneliti berpendapat bahwa MTBF merupakan pengukuran yang jauh lebih berguna daripada pengukuran cacat/KLOC. Secara sederhana dapat dikatakan bahwa seorang pemakai akhir lebih memperhatikan kegagalan, bukan jumlah cacat. Karena masing-masing cacat yangada pada sebuah program tidak memiliki tingkat kegagalan yang sama, maka penghitungan cacat total hanya memberikan sedikit indikasi tentang reliabilitas sistem. Contohnya adalah sebuah program yang telah beroperasi selama 14 bulan. Banyak cacat mungkin tidak terdeteksi dalam jumlah waktu yang lama sampai pada akhirnya cacat itu ditemukan. MTBF dari cacat yang tidak jelas seperti itu dapat berlangsung sampai 50, bahkan 100 tahun. Cacat yang lain, yang juga belum ditemukan, dapat memiliki tingkat kegagalan 18 atau 24 bulan. Meskipun setiap kategori pertama cacat (yang memiliki MTBF panjang) dihilangkan, pengaruhnya pada reliabilitas perangkat lunak tidak dapat diabaikan. Availabilitas perangkat lunak adalah kemungkinan sebuah program beroperasi sesuai dengan kebutuhan pada suatu titik yang diberikan pada suatu waktu dan didefinisikan sebagai: Availabilitas = MTTF / (MTTF + MTTR) x 100 % Pengukuran reliabilitas MTBF sama sensitifnya dengan MTTF dan MTTR. Pengukuran availabilitas jauh lebih sensitif daripada MTTR, yang merupakan pengukuran tidak langsung terhadap kemampuan pemeliharaan perangkat lunak

Studi kasus

Di dalam proyek pengembangan perangkat lunak, misalnya Anda akan membangun suatu aplikasi HRD berbasis web. Hal-hal yang perlu diperhatikan di dalam perencanaan kualitas adalah:

- Penentuan target yang ingin dicapai. Biasanya hal ini berhubungan dengan non functional requirement atau matrix, misalnya:
    - Availability, seberapa besar waktu suatu system untuk tersedia, misalnya 95% availability per bulan
    - Performance, seberapa cepat system ini dapat memproses transaksi per hari, biasanya ditentukan oleh TPS (Transaction Per Second)
    - Security, jenis security apa saja yang diperlukan
- Metode Pengujian yang akan dijalankan, misalnya:
    - Sistem Integration Test (SIT)
    - Performance Test
    - Security Test
    - User Acceptance Test (UAT)
- Environment apa yang akan digunakan untuk testing, apakah akan disediakan mesin khusus untuk testing atau pun menggunakan mesin development yang ada
- Alat bantu yang digunakan untuk testing, apakah perlu sewa atau beli, atau kan menyewa vendor lain untuk melakukan test
- Bagaimana penerimaan hasil test dibandingkan dengan perencanaan

BINUS UNIVERSITY ONLINE LEARNING

People Innovation Excellence

DAFTAR PUSTAKA

- Software engineering : a practitioners approach :
  Chapter 19/Pages 412 & chapter 20/Pages 431

- Introduction to software quality assurance,,
  http://www.youtube.com/watch?v=5_cTi5xBlYg

- Software reliability:
  http://www.youtube.com/watch?v=wv51aF_qODA

- Lean Six Sigma and IEEE standards for better software engineering,,
  http://www.youtube.com/watch?v=oCkPD5YvWqw

-