Pentingnya Menelaah Macam-Macam Kesalahan Rekayasa Perangkat Lunak

Ilham C. Suherman[1)], G. Y. Effendy[2)], Aufar I. Adianto[3)], Firin Handayani[4)], M. Arief Wujdi[5)], Nur Aini R[6)]
Departemen Sistem Informasi, Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
[1]ilham.cahya15@mhs.is.its.ac.id , [2]gregorius15@mhs.is.its.ac.id, [3]aufar.ilham16@mhs.is.its.ac.id,
[4]firin.handayani16@mhs.is.its.ac.id, [5]muchammad.arief.wujdi16@mhs.is.its.ac.id,
[6]nur.aini@is.its.ac.id

***Abstrak*— Dalam melakukan rekayasa dan pengembangan perangkat lunak, tim pengembang harus memastikan bahwa perangkat lunak yang dibuat benar-benar siap untuk diimplementasi atau diberikan kepada client. Oleh karena itu, tim pengembang harus bisa mengidentifikasi, mencegah, dan mengatasi berbagai macam kesalahan yang mungkin terjadi dalam proses pengembangan perangkat lunak. Artikel ini kami buat untuk memberikan kesadaran kepada tim pengembang, terutama tim pengembang yang belum berpengalaman, terhadap kesalahan-kesalahan yang telah terjadi dan mungkin akan terjadi pada saat mereka mengembangkan perangkat lunak. Agar kedepannya mereka dapat mengantisipasi dan mengambil langkah yang tepat ketika kesalahan tersebut benar-benar mereka alami.**

***Kata Kunci*— perangkat lunak, kesalahan, pengembangan, implementasi, antisipasi**

I. PENDAHULUAN

Rekayasa Perangkat Lunak atau yang biasa disebut sebagai *Software Engineering* merupakan penerapan atau penggunaan suatu pendekatan yang digunakan secara sistematis dan telah diuji secara kuantitas dalam mengembangkan, menggunakan, dan memelihara perangkat lunak. Menurut *IEEE Systems and Software Engineering Vocabulary,* Rekayasa perangkat lunak merupakan suatu penerapan sistematis dari pengetahuan teknologi, metode, dan pengalaman untuk mendesain, mengimplementasi, menguji, dan mendokumentasi suatu perangkat lunak [1]. Secara informal, rekayasa perangkat lunak dapat juga merupakan sebuah istilah untuk sekumpulan besar aktivitas untuk mengembangkan suatu perangkat lunak, mulai dari analisis, hingga perangkat lunak itu selesai diimplementasi [2].

Setiap tahapan yang dilakukan dalam proses pengembangan perangkat lunak, tidak lepasdari kesalahan-kesalahan yang mungkin diciptakan oleh pengembang. Dari kesalahan kecil dengan risiko rendah, hingga kesalahan besar dengan risiko tinggi. Kesalahan ini harus segera ditangani agar tidak terjadi masalah yang lebih besar di kemudian hari. Namun pada kenyataannya, banyak tim pengembang yang menyadari kesalahan yang mereka buat di pertengahan atau bahkan diakhir proses pengembangan sehingga resiko yang ditimbulkan semakin besar. Contohnya adalah adanya *bug* yang diakibatkan oleh sistematika pengujian program yang tidak baik, dan *bug* ini muncul ketika perangkat lunak akan diimplementasikan. *Bug* ini harus diatasi. Untuk mengatasinya, otomatis dibutuhkan penambahan waktu serta biaya yang dikeluarkan. Hal ini sangat berdampak bagi keberlangsungan proyek, dan seharusnya tidak terjadi jika tim pengembang dapat mengidentifikasi kesalahan ini diawal. Tujuan dari artikel ini adalah untuk meningkatkan *awareness* terhadap beberapa kesalahan yang sering terjadi dalam melakukan rekayasa dan pengembangan perangkat lunak sehingga untuk kedepannya diharapkan tim pengembang dapat menghindari kesalahan-kesalahan tersebut.

Artikel ini merangkum beberapa kesalahan yang sering terjadi pada saat pengembangan dan rekayasa perangkat lunak. Kesalahan-kesalahan ini diambil dari beberapa sumber, disertai dengan contoh studi kasusnya, .penyebab adanya kesalahan, serta cara penanggulangan atau cara penanganannya.

II. DISKUSI

Dalam setiap pengembangan perangkat lunak, melakukan pengujian menjadi salah satu kegiatan wajib yang perlu dilakukan. Ketika memutuskan proses pengujian yang tepat, pengembang akan selalu mempertimbangkan antara biaya, waktu dan kualitas. Waktu alat uji adalah salah satu faktor utama yang dipertimbangkan dalam menentukan biaya sistem pengujian [3]. Semakin lama proses pengembangan maka akan semakin banyak biaya yang dikeluarkan. Oleh karena itu penjadwalan akan sangat vital perannya di dalam proyek, terutama proyek TI.

Adanya cacat dalam sebuah perangkat lunak yang diperkenalkan dalam proses pengembangan akan selalu menghabiskan waktu dan uang untuk kedepannya. Kecacatan

membuat suatu sistem menjadi tidak efektif lagi, maka dari itu tidak jarang pengembang atau pengguna sistem perangkat lunak mengalami ratusan hingga miliyaran juta. Jika suatu sistem dikembangkan dengan buruk atau asal-asalan maka akan menyebabkan kemungkinan-kemungkinan buruk yang nantinya akan menyebabkan kerugian bukan hanya kepada pihak pengembang sistem melainkan terhadap pihak-pihak lain juga. Oleh karena itu dari kemungkinan buruk atau kerugian yang akan disebabkan di kemudian hari maka pengembangan perangkat lunak tidak bisa lagi dianggap remeh, dikarenakan fungsinya yang semakin kompleks. Berikut ini beberapa contoh kasus, yang terjadi dikarenakan tidak maksimalnya pengembang untuk melakukan pengujian sebelum perangkat lunak itu dijalankan atau digunakan.

A. Insufficient Testing

1) Kasus 1: Studi kasus yang berhubungan dengan pengembangan perangkat lunak permainan yang dikembangkan oleh Gearbox Software. Permainan ini bernama Aliens: Colonial Marines [4]. Pada awalnya, tidak ada yang mengetahui adanya *bug* pada aplikasi ini hingga ada seorang *modder* yang mencoba mengutak-atik *code* yang ada pada permainan tersebut. Setelah ditelusuri, *bug* ini diakibatkan oleh adanya kesalahan ketik pada program yang dijalankan. *Bug* ini sangat fatal karena menurunkan tingkat penjualan permainan. Kesalahan ini merupakan kesalahan dimana perangkat lunak yang telah dibuat / dikembangkan tidak melalui proses *testing* yang baik dan benar sehingga ketika sudah diimplementasi, banyak *end-user* yang mengeluhkan betapa banyaknya *bug* dan *glitch* yang ada pada perangkat lunak ini. Kesalahan ini dapat dicegah dengan cara menjalankan testing secara sistematis untuk memastikan bahwa perangkat lunak yang dibuat benar-benar bebas dari bug dan benar-benar siap untuk digunakan [5].

2) Kasus 2: Kasus Y2K dimana lebih dikenal dengan *Bug Millenium* yaitu suatu peristiwa yang terjadi di tahun 2000. Penyebab dari adanya kasus ini yaitu kesalahan perhitungan yang dilakukan oleh komputer dimana developer saat merancang sistem kalender ini hanya menyertakan 2 digit terakhir dalam rentan tahun yang berasumsi dua digit awal yaitu tahun"19-an". Sistem ini dirancang bertujuan untuk meminimalkan kapasitas penyimpanan dalam komputer [6].

Namun, kekhawatiran muncul, karena pada tahun 2000 terjadi perubahan tanggal pada komputer secara otomatis yaitu perubahan tanggal dari tahun 1999 menjadi tahun 1900. Kasus ini tidak banyak merugikan berbagai pihak karena banyak perusahaan yang langsung mengganti sistem perusahaannya karena ditakutkan akan terjadi hal-hal yang tidak diinginkan dan dari sistem diperbaiki oleh *progammer*. Meskipun banyak menghabiskan biaya untuk menyewa jasa *programmer,* namun langkah ini jauh lebih baik dan tidak merugikan banyak pihak.

3) Kasus 3: Pada tanggal 11 Desember 1998, NASA meluncurkan sebuah satelit bernama *Mars Climate Orbiter* dengan tujuan untuk mengenali iklim planet Mars. Sayangnya proyek senilai 327.6 juta USD ini hilang dan hancur akibat adanya kesalahan perhitungan yang disebabkan oleh perangkat lunak yang terpasang pada satelit. [7]

Perangkat lunak akan selalu berdampingan dengan perangkat keras dalam sebuah sistem, contoh kecilnya seperti software yang mengintegrasikan sistem komputer dengan sistem penyimpanan yang terhubung dengan tempat penyimpanan fisik. Disaat itulah perangkat lunak berinteraksi dengan perangkat lunak, dan disitu pula biasanya terjadi sebuah anomali. Kesalahan atau anomali tersebut terjadi pada dua sisi, bisa menjadi kesalahan ada pada perangkat lunak, maupun kesalahan pada perangkat keras. Banyak hal yang bisa terjadi saat interaksi tersebut.

Berikut beberapa contoh kesalahan saat interaksi keduanya yang disebabkan oleh perangkat lunak : *Buffer Overflow* yaitu kondisi dimana memori komputer lebih kecil dari yang diharapkan atau diperkirakan *programmer*, jadi selama pengoperasian sistem, salah satu program dalam sistem mengakses bagian yang salah dari memori komputer sehingga memungkinkan terjadi kecacatan*., Race conditions* yaitu peristiwa dimana waktu relatif spesifik dari berbagai komponen sistem menyebabkan perilaku yang tidak terduga. Kondisi seperti ini seringkali sulit dideteksi hanya dengan pengujian.

Dan berikut beberapa contoh kesalahan saat interaksi perangkat lunak dan perangkat keras yang disebabkan oleh perangkat lunak : Badly calibrated sensors yaitu kondisi dimana kalibrasi atau penyelarasan untuk sensors pada perangkat keras tidak dilakukan dengan baik, *Manufacturing / assembly process deficiencies* yaitu kegagalan dalam proses pembuatan atau perakitan, dimana bisa saja terjadi ada beberapa komponen yang tidak terpasang atau tidak sempurna saat proses perakitan, hal tersebut mengakibatkan kecacatan atau kegagalan interaksi antara perangkat lunak dan perangkat keras.

Selain itu, mengambil sebuah kasus mengenai kejadian meledaknya roket tak berawak Ariane 5 pada 4 Juni 1996 pada pukul 12:33 GMT (UTC) oleh *European Space Agency* yang mana meledak tepat setelah 40 detik setelah meluncur. Ariane 5 sendiri telah menelan banyak biaya sekitar $ 7 Miliar dan belum termasuk satelit didalamnya yang menelan biaya sekitar $ 100 Juta. Aplikasi SRI yang digunakan Ariane 5 sendiri sama dengan Ariane 4 pada 10 tahun sebelumnya. Ariane 5 gagal dikarenakan software SRI gagal mengkonversi data dari tipe data 64-bit menjadi 16-bit integer. Sebagai catatan bahwa 64-bit integer merupakan tipe data yang memiliki nilai lebih besar yang dapat diwakilkan oleh 16-bit integer yang mana mengakibatkan kelebihan beban dari SRI horizontal bias.

Kesalahan ini merupakan kesalahan spesifikasi yang mana spesifikasi yang digunakan tidak tepat dan modul yang sama serta tidak tepat digunakan kembali, serta seharusnya horizontal bias yang digunakan harus sesuai dengan 16-bit dan harus dinyatakan dalam bagian yang jelas didalam dokumen. Kesalahan ini dapat dicegah dengan cara menggunakan spesifikasi yang tepat dan tindakan yang tepat sebelumnya

melakukan sesuatu untuk memastikan bahwa kegagalan tidak akan terulang.

B. Estimating times too far out

Selanjutnya ada sebuah kasus Bandara seluas 140Km2 di Danver, Amerika Serikat yang merupakan bandara terbesar di Amerika Serikat. Dalam penggunaannya, sistem bagasi dilakukan secara otomatis, yang mana sistem akan menangani lebih dari 50 Juta penumpang tiap tahunnya. Maka dari itu sistem penanganan bagasi merupakan komponen yang penting dalam perencanaan tersebut. Namun dikarenakan manajemen bandara meremehkan kompleksitas proyek, mengakibatkan pembukaan bandara terhambat selama 16 Bulan dengan kerugian sekitar $ 528 Juta selama penundaan.

Saat pembukaan bandara dilakukan sistem ini hanya digunakan oleh satu maskapai penerbangan dan lainnya tetap menggunakan *belt conveyor*. Proses permintaan untuk desain dan konstruksi sistem tidak dimulai sampai dengan Musim Panas 1991 berdasarkan proyek asli maka hanya 2 tahun untuk kontrak perancangan dan pembuatan sebuah sistem tersebut. Dikarenakan waktu yang tidak mencukupi namun lingkup yang jauh lebih besar dan lebih kompleks, proyek tersebut menjadi jalur kritis didalam bandara demi memenuhi tanggal peresmian bandara yang telah direncanakan. Oleh karena itu, banyak masalah yang terjadi pada proyek oleh jalan pintas yang diambil tim untuk memenuhi jadwal yang mustahil

Kesalahan ini merupakan sebuah kegagalan dalam memperhatikan waktu yang ada didalam sebuah proyek *Denver International Airport* (DIA) yang mana dalam pelaksanaannya meremehkan kompleksivitas yang terlibat dalam perencanaannya dan sistem bagasi merupakan sistem yang paling kompleks dimana ukuran yang meningkat menghasilkan pertumbuhan yang eksponensial dalam kompleksifitas. Kesalahan ini dapat dicegah dengan memperhatikan waktu perencanaan pembangunan sebuah proyek agar sebuah sistem dapat terealisasikan dengan baik. Jika tidak melakukan perencanaan waktu yang baik, maka akan mengakibatkan banyak masalah dalam pelaksanaan proyek tersebut.

C. Error detection and handling are an afterthought and implemented through trial and error

Kesalahan ini salah satunya merujuk kepada sebuah studi kasus *Cancer Treatment* dimana terapi kanker yang menggunakan *software radiation.* Terapi ini dilakukan oleh Multidata Systems International dan terjadi di kota Panama pada tahun 2000 namun Software ini dikembangkan oleh perusahan Amerika Serikat. Kasus ini cukup menggemparkan di bidang medis sesudah adanya kasus yang serupa terjadi pada tahun sebelumnya yaitu kasus Therac-25 yang terjadi pada tahun 1985-1987 [8]

Kasus radiasi yang terjadi di kota Panama ini dikarenakan adanya kesalahan dalam aturan perhitungan dosis yang tepat untuk kegiatan terapi pasien yang menderita kanker tersebut. Kesalahan perhitungan tersebut terletak pada adanya perbedaan saat memasukkan data berdasarkan urutannya. Oleh karena itu, kesalahan tersebut sangat merugikan. Dimana terdapat 5 pasien yang meninggal karena mengalami overdosis. Kesalahan seperti ini dapat dihindari jika developer melakukan perhitungan secara tepat dan teliti sebelum membuat sebuah *software* [9]

D. Disconnected business priorities

Kasus Trilogy FBI yang dihentikan pada tahun 2005. Kasus ini berbeda dengan kasus lainnya dimana *software* sudah diimplementasi dan mendapat feedback. Kegagalan pada *software* ini ada saat pengembangannya selama 4 tahun, dimana proyek ini termasuk kategori proyek jangka panjang.

Tujuan sistem untuk FBI ini adalah untuk memungkinkan para agen berbagi file secara virtual tentang masalah ataupun informasi. Penyebabnya terletak saat pengembangannya, dimana terjadi kendala pada sistem yang dibuat kerena terlalu kompleks dan akhirnya sistem tersebut tidak bisa digunakan, sehingga harus dihentikan pengembangannya karena tidak mungkin untuk diimplementasi [10].

Permasalahan ini dapat dicegah dengan melakukan tiga pendekatan, yaitu dengan (a) mendiskripsikan strategi bisnis berdasarkan *balanced scorecard* dan menghubungkan infrastruktur perangkat lunak dengan indikator, (b) menggabungkan pendekatan *balanced scorecard* dengan ontologi perusahaan, dan (c) integrasi model bisnis dengan infratstruktur perangkat lunak[14].

E. Poor communications and documentations

1) Kasus 1: Kasus pembatalan pasport berhubungan dengan United Kingdom Passport Agency pada tahun 1999. Kasus ini terjadi karena adanya penggantian sistem lama dengan sistem baru. Proses antara kedua sistem berbeda khususnya bidang administrasi, namun karena tidak adanya pelatihan untuk para staff sehingga terjadi kebingungan yang menyebabkan produktivitas menurun meskipun adanya penambahan staff. Selain itu, tidak adanya pengujian yang tepat dan kurangnya dokumentasi terhadap sistem yang baru juga dapat menyebabkan kegagalan terhadap sistem sehingga *customer* banyak yang merasa dirugikan [12].

Dalam menerapkan sistem yang baru perlu adanya koordinasi antara satu dengan yang lainnya termasuk dalam hal komunikasi dan dokumentasi cara penggunaan sistem yang baru di suatu perusahaan. Hal ini terlihat jelas saat seseorang menganggap hal kecil atau sepele akan berdampak besar di akhir proses bisnis perusahaan. Peran dari para staff juga penting dalam hal penerapan sistem yang baru, dimana faktor manusia termasuk dalam komponen atau elemen yang penting dalam hal penerapan teknologi informasi. Selain staff yang tidak lain adalah *user*, yang menggunakan sistem tersebut. Detailnya, terdapat 6 komponen lainnya yaitu komunikasi data, basis data, jaringan komputer, perangkat lunak, perangkat keras, dan prosedur dalam implementasi sistem tersebut [13].

Jadi dalam hal ini, dokumentasi yang baik sangat berpengaruh terhadap jalannya proses bisnis. Dokumentasi yang baik akan mempermudah staff yang memakai *software*

yang dikembangkan. karena jika sistemnya sudah bagus namun yang mengoperasikan *software* tersebut tidak bisa menggunakannya, maka itu akan percuma saja. Staff yang baru tidak bisa serta merta langsung memahami dan mempraktekkan langsung sistem yang dimaksud sebelumnya dan akan memperlambat atau bahkan perusahaan tersebut akan mengalami kegagalan dalam penerapan sistem yang baru. Selain itu, dokumentasi yang baik juga dapat digunakan sebagai referensi dalam mengembangkan *software-software* lain yang mungkin akan dibutuhkan dimasa yang akan datang.

2) Kasus 2: Kompleksitas perangkat lunak IT yang tidak sesuai dengan bisnis yang dijalankan oleh *Child Support Agency* (CSA), menyebabkan kerugian besar terhadap perusahaan dan pihak-pihak yang terlibat di dalam maupun di luar [11]. EDS sebagai pihak yang mengembangkan dan mengusung sistem gagal memperkenalkan sistem terbaru tersebut terhadap CSA. Sebuah laporan mengatakan bahwa dampak dari kegagalan tersebut membuat para staf kebingungan mengoperasikan perangkat sehingga staf harus menjalankan proses bisnis secara manual akibat tidak dapat diandalkannya sistem.

Kesalahan ini terjadi karena EDS tidak mengkomunikasikan cara meggunakan perangkat lunak tersebut pada para staf [15].

III. KESIMPULAN

Tidak ada hal di dunia yang tak luput dari kesalahan, termasuk proses pengembangan perangkat lunak juga tidak luput dari berbagai macam kesalahan yang terjadi akibat kurangnya perhatian pengembang di tiap prosesnya. Akan tetapi, kesalahan-kesalahan tersebut dapat dicegah dan harus diminimalisir dengan berbagai cara pula agar tidak menimbulkan kerugian berbagai pihak terkait. Dalam pembahasan di atas dimana berbagai kesalahan yang dilakukan saat pengembangan perangkat lunak akan mengakibatkan dampak yang bermacam-macam pula.

Menurut sudut pandang kami mengenai apa yang telah dibahas maka dapat kami simpulkan bahwa kesalahan yang sering terjadi dalam pengembangan perangkat lunak yaitu saat tahapan testing atau pengujian. Hal itu dikarenakan saat melakukan pengujian terhadap sistem atau perangkat lunak sebelum didistribusikan dan dikonsumsi oleh masyarakat umum, tidak menutup kemungkinan jika sistem tersebut ada bug atau error yang akan terdeteksi. Meskipun demikian, saat sudah diketahui ada error dalam suatu sistem kebiasaan pengembang akan lebih memperhatikan penyebab terbesarnya. Namun, pada akhirnya penyebab kecil yang tidak disadari akan membawa dampak buruk di kemudian hari saat sudah dikonsumsi oleh publik sehingga hal itu akan sangat merugikan kedua belah pihak.

Adapun cara untuk mencegah hal-hal yang tidak diinginkan saat mengembangkan perangkat lunak secara umum yaitu sebagai berikut :

- Saat melakukan pengujian perangkat lunak harus diuji coba secara sistematis dan jika ditemukan bug maka langsung diatasi sehingga pengguna tidak merasa dirugikan
- Pengujian tidak hanya di satu perangkat keras melainkan di banyak hardware dengan spesifikasi yang berbeda. Hal ini karena bisa saja saat kita menggunakan perangkat keras A dengan spesifikasi b namun pengguna ada yang menggunakan perangkat keras B dengan spesifikasi yang berbeda pula.
- Dalam pengembagan suatu perangkat lunak jangan lupa untuk mendokumentasikan cara penggunaan dan berbagai hal mengenai perangkat lunak tersebut sehingga pengguna atau orang yang mengoperasikannya dapat mudah mengerti dan tidak menghambat proses bisnis.
- Dalam mengembangkan perangkat lunak alangkah baiknya jika kita sangat memperhatikan kebutuhan dan keinginan pengguna sehingga klien merasa terpenuhi dan pengguna puas dalam menggunakan perangkat lunak tersebut.

IV. REFERENSI

[1] *Systems and software engineering - Vocabulary*, ISO/IEC/IEEE std 24765:2010(E), 2010.

[2] Akram I. Salah (2002-04-05). "Engineering an Academic Program in Software Engineering" 35th Annual Midwest Instruction and Computing Symposium. Retrieved 2006-09-13.

[3] Antonia Bertolino, Eda Marchetti. "A Brief Essay on Software Testing" melalui https://www.coursehero.com/file/6370340/Bertolino-Marchetti-p393/ (diakses tanggal 23 September 2018)

[4] https://screenrant.com/aliens-colonial-marines-typo-broke-discovered/ (diakses tanggal 19 September 2018)

[5] Sandeep Dalal, Dr. Rajender Singh Chhillar, (2012-08-08). "Case Studies of Most Common and Severe Types of Software system Failure" International Journal of Advanced Research in Computer Science and Software Engineering, Halaman 344 dan Halaman 345, Volume 2

[6] https://techno.okezone.com/read/2017/08/04/207/1750186/tragedi-4-peristiwa-kesalahan-software-yang-disebabkan-oleh-bug (diakses tanggal 21 September 2018)

[7] https://www.intertech.com/Blog/15-worst-computer-software-blunders/ (diakses tanggal 22 September 2018)

[8] http://outfresh.com/knowledge-base/6-famous-software-disasters-due-lack-testing/ (diakses tanggal 21 September 2018)

[9] https://www.codepolitan.com/10-bug-software-dengan-dampak-yang-ekstrim-59a645ad3bbb5 (diakses tanggal 21 September 2018)

[10] http://aidaredo.scienceontheweb.net/index.php?option=com_content&view=article&id=51&Itemid=59 (diakses tanggal 21 September 2018)

[11] https://www.theregister.co.uk/2006/06/30/eds_csa/ (diakses tanggal 22 September 2018)

[12] https://www.nao.org.uk/report/united-kingdom-passport-agency-the-passport-delays-of-summer-1999/ (diakses tanggal 21 September 2018)
[13] http://lucky.blogstudent.mb.ipb.ac.id/2010/12/27/sistem-informasi-manajemen-sim-uat/ (diakses tanggal 22 September 2018)
[14] K. Sandkuhl, "Aligning Software Architecture and Business Strategy with Continuous Business Engineering," *Lecture Notes in Business Information Processing Advanced Information Systems Engineering Workshops*, pp. 14–26, 20
[15] https://www.computerweekly.com/feature/Inadequate-training-poor-communication-and-IT-problems-are-to-blame say-CSA-staff (diakses tanggal 22 September 2018)