Jurnal Mantik Penusa
Volume 4, No. 1, Juni 2020
e-ISSN 2580-9741
p-ISSN 2088-3943

EFEK BEHAVIOR-DRIVEN DEVELOPMENT, HEXAGONAL ARCHITECTURE DAN EVENT SOURCING TERHADAP PENGEMBANGAN PERANGKAT LUNAK BISNIS

[1)]Muhamad Biyan Abdulah, [2)]Agung Triayudi, [3)]Andrianingsih

[1]Sistem Informasi, Fakultas Teknologi Komunikasi dan Informatika, Universitas Nasional, Jl. Sawo Manila, Pejaten
Ps. Minggu, Jakarta 12520, Indonesia
E-mail : biyandonk@yahoo.co.id
[1]Sistem Informasi, Fakultas Teknologi Komunikasi dan Informatika, Universitas Nasional, Jl. Sawo Manila, Pejaten
Ps. Minggu, Jakarta 12520, Indonesia
E-mail : agungtriayudi@civitas.unas.ac.id
[1]Sistem Informasi, Fakultas Teknologi Komunikasi dan Informatika, Universitas Nasional, Jl. Sawo Manila, Pejaten
Ps. Minggu, Jakarta 12520, Indonesia
E-mail : andrianingsih@civitas.unas.ac.id

*)Email Penulis Korespondensi : agungtriayudi@civitas.unas.ac.id

ABSTRACT

Software usage in business world are growing. But as time passes, many problems occurs as both software users and software features that wished to be developed increases, such as performance problems and maintainibility problems. Theres some techniques that has been developed to solve these kinds of problem (such as MVC framework, Test-Driven Development, Domain-Driven Design) but sometimes these techniques criticized as adding more problems instead. In this journal we will present our findings from our research which utilizes two techniques that is Behavior-Driven Development and Event Sourcing and observes its effects in our"Assignment Management System" we've developed.

Keywords: *behavior-driven development; event sourcing; business application; software development process; unit testing; test-driven development*

Abstrak

Penggunaan perangkat lunak dalam dunia bisnis makin diminati. Namun seiring waktu, banyak terjadi permasalahan yang seringkali terjadi seiring meningkatnya jumlah pengguna dalam skala besar dan fitur-fitur yang ingin dikembangkan, seperti permasalahan performa dan tingkatan kemampuan perangkat lunak untuk dirawat seiring waktu. Banyak teknik yang dikembangkan untuk menangkal hal ini (seperti kerangka kerja MVC, Test-Driven Development, Domain-Driven Design) namun kadangkala teknik tersebut dikritik menambah lebih banyak masalah. Pada jurnal ini kami akan meneliti dua teknik yang dikembangkan untuk mempermudah pengembangan perangkat lunak yaitu Behavior-Driven Development dan Event Sourcing dan meninjau efeknya terhadap Sistem Manajemen Surat Tugas yang kami kembangkan.

Kata Kunci: *behavior-driven development; event sourcing; aplikasi bisnis; proses pengembangan perangkat lunak; unit testing; test-driven development*

1. PENDAHULUAN

Penggunaan komputer untuk mempermudah urusan-urusan yang dimiliki manusia telah berlangsung lama dimulai dari sekadar angan-angan untuk membuat mesin yang dapat melakukan ragam perhitungan[1], keperluan militer[2] hingga keperluan bisnis skala besar[3].

Digitalisasi bisnis bukan hanyalah sekadar hal yang harus dilakukan namun juga hal yang penentu hidup atau matinya perusahaan bisnis modern. Namun, di satu sisi, Gregor Hohpe menjelaskan bahwa pandangan digitalisasi bisnis dapat dibagi menjadi dua, yaitu berbasis "Economies of Scale" dan "Economies of Speed", dimana perusahaan berbasis "Economies of Scale" sekadar menambahkan akses digital terhadap bisnis yang sudah ada yang terwujud dalam bentuk project-based yang biasanya dimana perusahaan berbicara dalam hal target waktu, biaya dan tujuan yang terbatas dan juga menganggap bahwa "setelah projek selesai, kehidupan perusahaan akan kembali berjalan normal seperti sedia kala" dan dimana

perusahaan berbasis "Economies of Speed" terus mengubah model bisnis mengikuti peluang-peluang yang dihadapkan yang salah satunya adalah peluang kemampuan yang ditawarkan oleh dunia digital dan lebih menganggap bahwa "perubahan konstan = normal". Contoh dampak negatif perusahaan yang gagal beradaptasi karena kakunya model bisnis yang dimiliki di Indonesia sendiri dapat dilihat dari tergilasnya perusahaan taksi yang telah berdiri sejak lama oleh perusahaan ojek online yang bergerak cepat dan hadirnya toko online. Sejumlah perusahaan dikabarkan telah beralih menggunakan pandangan "Economies of Speed" seperti IAG dan Bank ANZ[4][5]. "Economies of Speed" juga berkaitan sifat ketangkasan (agility), dimana dalam dunia pengembangan perangkat lunak, terinsipirasi dari proses manufaktur tangkas yang dibuat Toyota untuk mengurangi buangan sia-sia, diciptakanlah "Manifesto Agile" serta ragam metode penerapannya seperti Scrum dan Kanban.

Dalam dunia perangkat lunak, walaupun sumber kode program bukanlah produk utama yang ditampilkan ke pengguna, tingkat kerapihan sumber kode program dianggap penting dan menjadi faktor penentu terhadap kecepatan pengembangan perangkat lunak, yang asal mulanya dapat dirujuk hingga tahun 1968 dimana Edsger W. Dijkstra mengutarakan bahwa pernyataan GOTO pada program dianggap membawa dampak buruk. Pada bukunya, Robert C. Martin mengutarakan bahwa seiring waktu, pengembang perangkat lunak dimudahkan oleh tools yang ada (bahasa pemrograman, compiler dan, lain sebagainya) dengan cara "dihilangkan sebagian kendalinya atas sesuatu" (seperti contoh tadi, dimana GOTO mulai hilang dari bahasa modern sebagai wujud dihilangkannya kendali pemrogram untuk melompat dari satu pernyataan ke pernyataan lain saat program tengah berjalan). Selain itu, beberapa prinsip lain juga dihadirkan yang bila diikuti, akan menjaga tingkat kerapihan program, seperti prinsip S.O.L.I.D yang merupakan singkatan dari[6]: Single Responsibility Principle, Open/Closed Principle, Liskov Substitution Principle, Interface Segregation Principle dan Dependency Inversion Principle.

Ragam kerangka kerja, pustaka dan prinsip-prinsip dihadirkan oleh beberapa pihak dan mengklaim dapat mempermudah kerja pemrogram. Untuk kerangka kerja sendiri terdiri baik dari yang sudah lama berdiri dan mengklaim bahwa kerangka kerja buatannya telah sukses digunakan perusahaan besar dan telah sukses mempermudah kerja pemrogram[9] maupun dari komunitas Javascript akhir-akhir ini[7][8]. Namun, ditemukan bahwa kerangka kerja, pustaka dan prinsip-prinsip yang dipakai banyak kalangan seringkali justru melanggar prinsip-prinsip yang membuat suatu kode program rapih. Kerangka kerja (yang juga sekaligus prinsip) MVC dipandang hal yang buruk jika tidak memberikan transparansi kendali yang jelas yang mengakibatkan program tidak fleksibel untuk dibentuk[6]. Beberapa prinsip, seperti prinsip Anemic Domain Model dari prinsip Domain Driven Design, dianggap justru menyalahi prinsip yang dapat membuat program rapih[10]. Dalam dunia perangkat lunak, prinsip yang diniatkan untuk membantu namun berakhir menambah masalah disebut sebagai "Anti-Pattern"[11].

Dari ragam prinsip yang ada baik yang telah disebutkan sebelumnya seperti MVC dan Domain Driven Design maupun yang tidak seperti Model Driven Architecture, Test Driven Development, MVP dan MVVM, kami akan meneliti apa dampak Behavior Driven Design, Hexagonal Architecture dan Event Sourcing terhadap perangkat lunak yang kami akan kembangkan.

Behavior-Driven Development adalah prinsip pengembangan perangkat lunak yang dikembangkan dari Test-Driven Development, yang pertama kali dirilis oleh Dan North dalam bentuk tool bernama JBehave (yang lalu lanjut dikembangkan menjadi tool bernama Cucumber). Behavior-Driven Development memiliki kesamaan mencolok bila dibandingkan dengan pendahulunya Test-Driven Development, yakni proses pengembangan program dimulai dari dimulainya perancangan Unit Test terotomatisasi yang dapat dijalankan secara cepat. Perbedaannya adalah Behavior-Driven Development cenderung dimulai dengan proses perancangan berkas fitur yang terdiri dari ragam skenario dan ragam langkah uji yang biasa dibagi menjadi tiga kata kunci yakni Given, When dan Then.[12]

Adapun Hexagonal Architecture adalah prinsip dimana use cases bisnis diwujudkan dalam bentuk abstrak dan implementasi (disebut sebagai Ports and Adapter). Hal ini menjadikan program fleksibel dan tidak terkekang satu implementasi fitur.[6]

Event Sourcing adalah prinsip dimana tiap action aplikasi direkam dalam sebuah Event, disimpan dan apabila dibutuhkan state aplikasi dapat dibuat dari nol dari kumpulan Event yang telah direkam. [13]

2. METODE

Aplikasi yang dibuat kali ini adalah aplikasi manajemen surat tugas yang mencontohkan sebuah perusahaan yang rutin membuat surat tugas yang menugaskan para pegawainya. Tiap surat tugas tersebut memiliki batas waktu pelaporan. Pegawai yang ditugaskan harus mengunggah laporan tersebut sebelum batas waktu yang ditentukan. Apabila pegawai tersebut terlambat mengunggah surat tugas tersebut, maka sistem akan mewarnai

surat tugas tersebut dengan warna merah.

Penelitian akan dimulai dengan menerapkan sistem tersebut dengan Behavior-Driven Development, Hexagonal Architecture dan Event-Sourcing. Setelah sistem diterapkan, pengaruhnya akan ditinjau berdasarkan struktur class source code tersebut dan performa runtimenya.

Mengingat pada penelitian kali ini aplikasi yang dikembangkan akan dikembangkan dengan menggunakan tahapan Behaivor Driven Development, maka pengembangan aplikasi dapat dimulai dengan membuat "Given-When-Then" scenario fitur utama aplikasi tersebut. Langkah "Given-When-Then" pada aplikasi kali ini adalah

1. Jika sekarang tanggal: 1 Januari 2020
2. Jika user berjumlah 10 orang
3. Jika Admin adalah "User 1"
4. Jika Pembuat Surat Tugas adalah "User 2"
5. Jika Sesjen adalah "User 3"
6. Jika Irjen adalah "User 4"
7. Jika dibuatkanlah Surat Tugas berupa menugaskan: "User 5", "User 6", "User 7" pada tanggal: 6 Januari 2020 dan berdurasi: 2 hari
8. Jika Sesjen dan Irjen menyetujui
9. Maka Surat Tugas akan berwarna Putih
10. Jika sekarang tanggal: 9 Januari 2020
11. Maka Surat Tugas akan berwarna Merah
12. Jika "User 5" upload laporan
13. Maka Surat Tugas akan berwarna Merah
14. Jika sekarang tanggal: 10 Januari 2020
15. Jika "User 6" upload laporan
16. Jika "User 7" upload laporan
17. Maka Surat Tugas akan berwarna Kuning
18. Jika sekarang tanggal: 11 Januari 2020
19. Jika Sesjen dan Irjen menyetujui laporan tersebut
20. Maka Surat Tugas akan berwarna Hijau

Untuk mengikuti alurkerja Hexagonal Architecture, aplikasi selanjutnya diterapkan dengan menjabarkan abstraksi kemampuan-kemampuan aplikasi tersebut. Abstraksi kemampuan-kemampuan tersebut, bila bahasa pemrograman yang digunakan menggunakan paradigma OOP, dapat dijabarkan dengan abstract class dengan seluruh isi property dan method terkaitnya. Sedangkan dalam bahasa yang menggunakan paradigma functional, abstraksi kemampuan tersebut dapat dijabarkan dengan ragam type alias. Setelah abstraksi kemampuannya selesai, dibuatlah implementasi kemampuannya.

Implementasi yang pertama kali dilakukan adalah implementasi menggunakan "Test Double", yang merupakan praktek yang memang disarankan. Implementasi menggunakan "Test Double" dapat mempersingkat waktu pengerjaan sekaligus mempersingkat waktu pengujian secara otomatis.

Setelah implementasi selesai, dapat dibuatkanlah pula implementasi pembungkus yang membungkus implementasi yang diuji dan juga implementasi yang mencatat event-event yang terjadi yang akan ikut terpanggil.

Event-event yang dicatat antara lain event pada server dan event pada client. Pada tiap pencatatan event akan direkam pula waktu pelaksanaannya.

3. HASIL DAN PEMBAHASAN

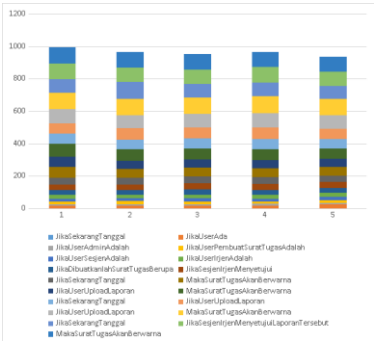
Behavior Driven Development dan Hexagonal Architecture terlihat membantu dalam pengembangan. Dalam proses pengembangan, bila menggunakan bahasa pemrograman static typed, IDE akan memperlihatkan apakah implementasi kemampuan yang ada telah mencocoki abstraksi kemampuan yang ada atau tidak.

Menggunakan Event Sourcing dapat pula dilihat kecepatan saat pengujian dilaksanakan. Berikut adalah hasil yang didapat saat pengujian otomatis dilaksanakan dengan menggunakan implementasi client dan server dalam bentuk Test Double.

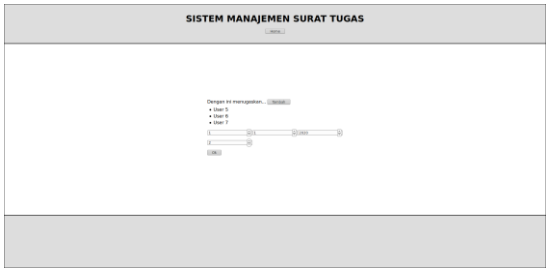
TABLE 1.
Tabel uji performa tiap percobaan (menggunakan Test Double)

Langkah	Percobaan				
	1	2	3	4	5
JikaSekarangTanggal	0	1	1	1	**1**
JikaUserAda	16	14	16	16	**25**
JikaUser AdminAdalah	8	10	8	10	**9**
JikaUser PembuatSuratTugasAdalah	16	19	17	16	**16**
JikaUser SesjenAdalah	19	18	20	17	**18**
JikaUserIrjenAdalah	23	21	23	24	**26**
JikaDibuatkanlahSuratTugasBerupa	31	30	34	31	**31**
Jika SesjenIrjen Menyetujui	34	34	36	37	**36**
JikaSekarangTanggal	41	40	40	39	**39**
Maka SuratTugas Akan Berwarna	67	56	55	55	**56**
Jika UserUpload Laporan	64	52	53	53	**48**
Maka SuratTugas Akan Berwarna	79	71	67	67	**63**
JikaSekarangTanggal	62	59	61	61	**6**

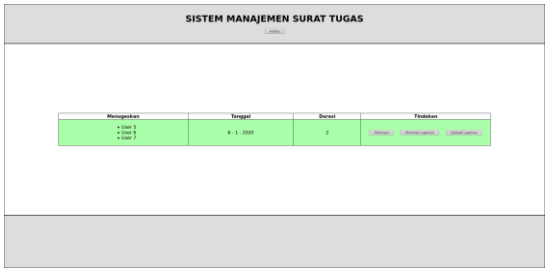
					0
JikaUser Upload Laporan	66	68	67	70	**64**
JikaUser Upload Laporan	87	83	84	91	**83**
MakaSurat TugasAkan Berwarna	101	99	103	105	**99**
JikaSekarangTanggal	84	104	83	84	**82**
JikaSesjen Irjen Menyetujui Laporan Tersebut	95	88	88	94	**87**
MakaSurat TugasAkan Berwarna	**101**	**96**	**95**	**95**	**94**



GAMBAR 1.
Hasil catatan waktu pelaksanaan per step yang dilakukan saat pengujian berlangsung (menggunakan Test Double; selisih antara Start dan End) dari satu percobaan di satu waktu ke percobaan di satu waktu yang lain. Satuan waktu dalam milidetik. Lebih kecil lebih baik.



GAMBAR 2.
Tampilan Sistem pada halaman pembuatan Surat Tugas



GAMBAR 3.
Tampilan Sistem pada halaman daftar Surat Tugas yang membutuhkan perhatian User. Dapat di lihat baris berwarna hijau menandakan urusan dengan Surat Tugas telah selesai

4. KESIMPULAN

Behavior-Driven Development dan Hexagonal Architecture membantu pemrogram dalam perjalanannya mengembangkan sebuah aplikasi bisnis. Tak hanya IDE akan lebih membantu mengingatkan pemrogram terhadap method-method apa saja yang harus dibuat saat implementasi (Compiler-Driven Development) tapi dari segi struktur source code. Dari satu implementasi ke implementasi yang lain, letak file yang berubah adalah sama dan tiap struktur lebih memperlihatkan tanggung jawabnya sendiri masing-masing (struktur tampilan hanya berisi tampilan sedangkan struktur business logic hanya berisi business logic), mematuhi prinsip SOLID yang sebelumnya dijelaskan.

Event Sourcing dapat membantu pengujian performa aplikasi dan dalam penelitian kali ini terlihat bahwa menggunakan Test Double (selain memperjelas source code) tidak terlalu berpengaruh buruk terhadap performa di mana walaupun performa tidak sepenuhnya determinstik kendati step yang dilakukan sama namun waktu pelaksanaannya tidak terlalu buruk (kurang lebih 1 detik). Ini mengikuti prinsip bahwa “unit testing harus dilakukan secara cepat” yang sebelumnya oleh Kent Beck dan Robert C. Martin tulis dalam bukunya.

5. DAFTAR PUSTAKA

[1] Halacy, Daniel Stephen (1970). Charles Babbage, Father of the Computer. Crowell-Collier Press. ISBN 978-0-02-741370-0.
[2] Welchman, Gordon (1997) [1982], The Hut Six story: Breaking the Enigma codes, Cleobury Mortimer, England: M&M Baldwin, p. 81, ISBN 978-0-947712-34-1
[3] Brooks, Frederick P.; Iverson, Kenneth E. (1963). Automatic Data Processing. Wiley. hal. 94 "semiautomatic".

[4] The Architect Elevator – Visiting the upper floors, https://martinfowler.com/articles/architect-elevator.html
[5] Products Over Projects, https://martinfowler.com/articles/products-over-projects.html
[6] Clean Architecture: A Craftsman's Guide to Software Structure and DesignSeptember 2017, Robert C. Martin, ISBN:978-0-13-449416-6
[7] Yet Another Framework Syndrome, https://medium.com/tastejs-blog/yet-another-framework-syndrome-yafs-cf5f694ee070
[8] You Don't Need a JavaScript Framework, https://medium.com/better-programming/you-dont-need-a-javascript-framework-df2a36c2dd0a
[9] Ruby on Rails, https://rubyonrails.org/
[10] The Anaemic Domain Model is no Anti-Pattern, it's a SOLID design, https://blog.inf.ed.ac.uk/sapm/2014/02/04/the-anaemic-domain-model-is-no-anti-pattern-its-a-solid-design/
[11] Koenig, Andrew (March–April 1995). "Patterns and Antipatterns". Journal of Object-Oriented Programming. 8 (1): 46–48.;
[12] Introducing BDD, https://dannorth.net/introducing-bdd/
[13] Why use Event Sourcing?, http://codebetter.com/gregyoung/2010/02/20/why-use-event-sourcing/