AGILE METHODOLOGY : REFLECTION FROM TESTING SIDE

Muhamad Adham Hambali
adhamhambali@kuis.edu.my

Ahmad Nazeer Zainal Arifin
ahmadnazeer@kuis.edu.my

ABSTRAK

Metodologi Agile semakin mendapat tempat di syarikat perisian lokal mahupun antarabangsa bagi menggantikan metodologi klasik Waterfall. Metodologi agile hakikatnya lebih bersifat insani dengan mementingkan gerak kerja berpasukan. Namun begitu proses transformasi ke metod agile berdepan kesukaran kepada pekerja perisian yang sudah biasa dengan metod waterfall terutama mereka yang berada dalam fasa verifikasi perisian. Kajian ini menfokuskan tentang pengujian regresi perisian dalam transformasi Metodologi Scrum Agile dari Metodologi Waterfall menggunakan gabungan kaedah kajian perpustakaan dan pemerhatian umum penulis sepanjang berada dalam industri pembangunan perisian.

Kata kunci*: Metodologi Scrum Agile, Pengujian Regresi Perisian, Waterfall*

ABSTRACT

Agile methodology is gaining ground in local and international software companies to replace Waterfall Classic Methodology. The agile methodology is in fact more humanistic with the emphasis on teamwork. However, the process of transforming to agile methodology is difficult for software workers who are familiar with the waterfall method, especially those in the software verification phase. This study focuses on software regression testing in the transformation of Agile Scrum Methodology from Waterfall Methodology using a combination of library research methods and the author's general observation in software development industry.

Keywords*: Scrum Agile Methodology, Software Regression Testing, Waterfall*

Pendahuluan

Metodologi *waterfall* ialah model pembangunan perisian pertama yang diperkenalkan oleh Dr. Winston W. Royce (1970). Walaupun istilah *waterfall* tidak dinyatakan Royce di dalam artikelnya *Managing The Development Of Large Software Systems*, istilah *waterfall* kemudiannya disebut di dalam artikel oleh T.E. Bell dan T. A. Thayer (1976). Ia difahami dengan baik oleh Bell dan Thayer dengan metodologi klasik itu menjadi model utama kepada pembangunan perisian sebelum munculnya metodologi *agile* pada awal 2000.

Kemunculan metodologi *agile* tidaklah terus membunuh metodologi klasik *waterfall*. Kebanyakan syarikat masih menggunakan kaedah klasik sehinggalah para pembangun dan penguji perisian faham dan mahir dengan kaedah *agile*. Banyak latihan-latihan disediakan oleh syarikat-syarikat gergasi dengan mengambil kira faktor penjimatan kos dan keuntungan untuk jangka masa panjang apabila *agile* dilaksanakan. Banyak projek perisian boleh diterima masuk ke dalam senarai projek tahunan syarikat berbanding kaedah klasik. Kaedah *agile* juga lebih telus dengan mengambil kira pandangan semua *stakeholders* semasa pembangunan perisian masih diperingkat merangka sehingga fasa implementasi. Tim Brizard (2017) menerangkan bahawa kaedah-kaedah dalam *Agile Scrum* misalnya memperlihatkan usaha kearah ketelusan yang tinggi seperti aktiviti *daily standup*, *sprint planning*, *sprint review* dan mesyuarat *retrospective.*

Namun begitu, ia tidaklah mudah untuk dilaksanakan dengan kaedah klasik *waterfall* sudahpun sebati dengan sebahagian besar pembangun perisian. Kaedah *waterfall* memberi ruang yang besar untuk pembangun perisian merancang projek perisian dengan langkah berhati-hati yang tinggi diambil bagi setiap fasa memandangkan kaedah ini dijalankan secara linear dan berurutan. Pendek kata, dengan kaedah ini setiap fasa hendaklah disemak dan disahkan oleh Pengurus Projek sebelum berpindah ke fasa seterusnya. Ia mengambil masa lebih panjang dan lama, maka banyak projek perisian tidak boleh dibangunkan dalam masa setahun berbanding kaedah *agile*. Justeru, kaedah *agile* menjadi pilihan kebanyakan syarikat masa kini yang bukan sahaja mementingkan kualiti produk perisian yang dihasilkan malah keuntungan juga menjadi agenda utama untuk syarikat-syarikat ini kekal bersaing.

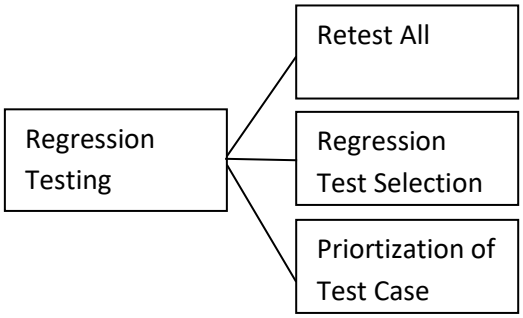
Pengujian Regressi Dan Kepentingannya Dalam Fasa Pengujian

Menurut Pezze, Mauro, Young, Michal (2008), pengujian regressi ialah menjalankan semula ujian fungsi dan bukan fungsi perisian demi memastikan perisian yang sudah dibangunkan dan diuji berjalan seperti biasa selepas perubahan ke atas perisian tersebut dilakukan. Apa yang menjadikan pengujian regressi lebih penting yang mesti dijalankan ialah bagi memastikan penambahan yang dilakukan ke atas perisian tidak mengubah fungsi sedia ada. Jika ia berlaku kos penyelenggaraan akan bertambah kerana perisian harus dibaiki dari awal dengan mengenal pasti ‘root cause’ kepada masalah.

Mengenal pasti punca utama kerosakan atau hilang fungsi asal perisian sangat mencabar. Sebelum menyerahkan kepada pembangun perisian untuk menyelesaikan kerosakan yang berlaku, penguji perisian yang sedang menjalankan aktiviti pengujian harus mencari puncanya terlebih dahulu. Ada kerosakan yang mengambil masa sehari atau lebih untuk dikenal pasti puncanya. Ada kalanya disebabkan kesilapan penguji perisian itu sendiri yang tidak mengemaskini elemen-elemen baru sebelum sesi pengujian dilaksanakan. Ada kalanya disebabkan data uji yang bermasalah dan bertindan. Semua ini harus diperiksa silang berkali-kali sebelum diserahkan kepada pembangun perisian untuk dibetulkan.

Dalam metodologi *waterfall* sebagai contoh, (Alexandar Sergeev, 2016) menyatakan peringkat pengujian perisian kebiasannya dilakukan ketika di peringkat akhir pembangunan dan hanya dikendalikan oleh kumpulan penguji perisian sahaja. Ini bermakna fasa pengkodan hanya dikendalikan oleh kumpulan pembangun perisian sahaja. Walaupun tidak terlibat secara langsung dengan sesi pengkodan, penguji perisian harus faham dan mahir membaca kod pengaturcaraan supaya apabila berlaku *abnormal end* atau *error* ketika sesi pengujian perisian dijalankan mereka faham apa yang sedang berlaku dan boleh juga terlibat membantu pembangun perisian ketika sesi pulih perisian.

Rajah 1: Kaedah Pengujian Regressi



Sumber: http://www.guru99.com

Menurut guru99.com, pengujian regressi boleh dilakukan melalui tiga (3) metod iaitu pengujian semula semua (*retest all*), pengujian regressi terpilih (*regression test selection*) dan keutamaan kes-kes pengujian (*prioritization of test cases*). Metod pengujian semula semua bermaksud melakukan ujian semula ke atas semua kes-kes pengujian. Jadi melakukan pengujian semula ke atas semua kes-kes bererti ia metod yang mahal berbanding metod-metod yang akan dibincangkan lebih lanjut. Metod ini memerlukan masa dan perbelanjaan yang tinggi.

Metod pengujian regressi terpilih menjadi pilihan setelah metod pertama memerlukan masa yang lama dan perbelanjaan yang tinggi. Metod ini memerlukan penguji untuk memilih sebahagian kes-kes dari set pengujian yang akan dijalankan untuk pengujian regressi. Metod ini lebih menjimatkan kos dan masa memandangkan ujian dilakukan ke atas kes-kes penting terpilih sahaja.

Metod ketiga iaitu keutamaan kes-kes pengujian pula kaedah yang mengutamakan kes-kes pengujian yang bergantung kepada impak perniagaan, kritikal dan kekerapan penggunaan fungi-fungsi. Pemilihan kes-kes pengujian berdasarkan keutamaan akan dapat mengurangkan set pengujian dengan lebih berkesan.

Definisi Mainframe

Menurut techopedia.com, mainframe ialah komputer yang bersaiz besar, berkuasa tinggi dengan ruang simpanan yang banyak dan mempunyai kebolehpercayaan tahap tinggi. Mainframe telah lama wujud dan menjadi dominan bermula 1950an sehingga 1970an. Mainframe digunakan dengan tujuan yang berskala besar yang memerlukan kebolehdapatan dan keselamatan yang hebat. Ed Scannell (2017) menyebut dalam artikelnya "ia mencipta namanya lama dahulu, namun begitu, walaupun di masa perkomputeran 'virtualization' dan 'cloud' (wujud), mainframe masih ada."

Pengujian Perisian Dalam Mainframe

Pengujian perisian dalam *mainframe* boleh didefinisikan sebagai pengujian sistem *mainframe* itu sendiri. Ia seakan-akan pengujian berasaskan laman (*web based testing*) tetapi lebih sukar kerana melibatkan kemahiran teknikal tertentu. Aplikasi mainframe (dipanggil *job batch*) diuji berpandukan *test cases* yang dibuat berdasarkan keperluan sistem yang perlu kepada pengujian.

Pengujian dalam *mainframe* selalunya dilaksanakan ke atas *deployed code* menggunakan pelbagai kombinasi data dan disediakan dalam *input file*. *Deployed code* bermaksud ianya berfungsi dan bersedia untuk digunakan dalam pasaran. Jikalau pengaturcara menulis 25000 baris kelas tanpa *error* dan berfungsi dengan baik ia tetap dikira berjaya. Tidak kira jika ia nampak teruk dan buruk. Jika ia berfunsgi, ia berfungsi.

Penyediaan *input file* amat penting sebelum aktiviti pengujian (*batch job testing*) dilakukan. *Input file* yang tidak mempunyai data tidak akan menghasilkan keputusan yang dikehendaki melalui laporan dan *output file*. Selain cara manual dengan memasukkan data pengujian ke dalam *input file*, cara yang lebih pantas untuk mengetahui keputusan pengujian ialah dengan melakukan pengujian secara masa sebenar (*real time*) atau pengujian atas talian (*online testing)* dengan memasukkan data ke dalam satu program (*online cics*) di mana keputusan akan diberikan secara masa sebenar juga. Ia seperti pergi ke *ATM* untuk memeriksa baki, dan permintaan dihantar ke satu program yang menerima permohonan pemeriksaan baki

dan memulangkan permohonan secara masa sebenar. Jika berjaya, program akan memberitahu baki terkini dan jika tidak keputusan adalah sebaliknya diperolehi.

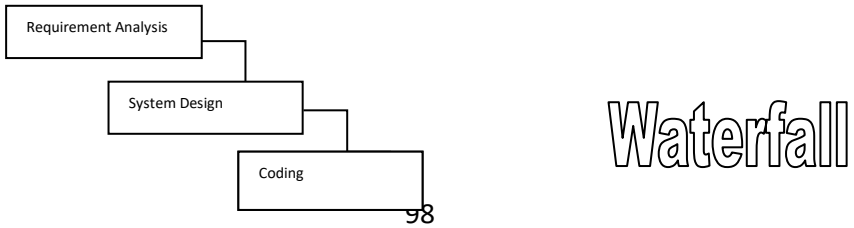
Pengujian *batch job* melibatkan aktiviti menjalankan beberapa *batch jobs* bagi fungsi yang sudah dilaksanakan dalam *current release*. *Batch jobs* boleh dijalankan secara manual dengan menggunakan *command* khas SUBMIT atau SUB sahaja. *Batch jobs* yang sedang berjalan boleh dipantau dengan menggunakan *command* STATUS. Ada beberapa lagi *command* yang akan dibincangkan kemudian. Setelah *batch job* tertentu berjaya dijalankan dan berakhir dengan baik, keputusan ujian boleh diekstradisi dari *output file* dan pengkalan data. Keputusan tersebut hendaklah disemak dan didokumenkan.

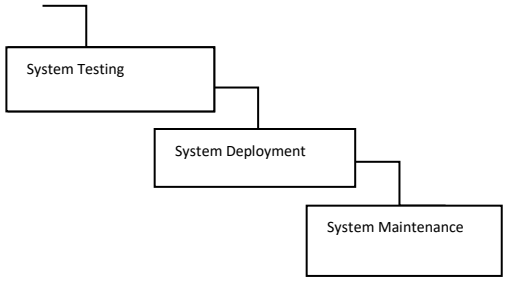
Pengujian atas talian (*online testing)* pula merujuk kepada pengujian ke atas skrin-skrin CICS. CICS bermaksud *Customer Information Control System* atau dalam bahasa melayu, sistem kawalan maklumat pelanggan. Ia adalah sebuah platfom untuk memproses sebarang aktiviti transaksi secara atas talian. Ia hampir sama dengan pengujian ke atas laman sesawang tetapi CICS berfungsi di dalam aplikasi sistem *mainframe.* Fungsi bagi skrin-skrin CICS sediaada boleh berubah dan skrin-skrin baru boleh ditambah dari semasa ke semasa. Fungsi-fungsi skrin CICS inilah yang perlu diuji sebagai sebahagian dari pengujian atas talian (*online testing).*

Ulasan Ringkas Mengenai Pengujian Regressi Dalam Waterfall

Seperti yang dibincangkan sebelum ini, metodologi *waterfall* dijalankan secara linear atau berurutan. Setiap peringkat mesti disempurnakan sebelum berpindah ke peringkat seterusnya. Kesempurnaan sesuatu peringkat pembangunan perisian ditentukan apabila kriteria masuk dan keluar (*entry and exit criteria*) dipenuhi dan kemudiannya disahkan oleh pengurus yang bertanggungjawab dan pengurus projek (*Line Manager and Project Manager*).

Rajah 2: Metodologi Waterfall





Peringkat-peringkat dalam metodologi *waterfall* boleh berubah atau ditambah terutamanya diperingkat awal pembangunan perisian. Peringkat Kajian kebarangkalian atau *feasibility study* boleh juga dilakukan sebelum peringkat analisis keperluan atau *requirement analysis.* Ia bergantung kepada kehendak perniagaan bagi sesuatu produk baru (*new product*) atau pemanjangan kepada produk sedia ada (*product enhancement*). Ia selalunya penting bagi mengetahui maklum balas terhadap cadangan produk yang hendak dibangunkan samaada ia praktikal ataupun sebaliknya.

Ketika menjalankan pembangunan projek, antara perkara pokok yang akan dibincang dan dimasukkan ke dalam perancangan pengujian ialah pengujian regressi. Pengujian regressi dalam metodologi tradisional *waterfall* boleh dilakukan ketika SIT (*System Intergration Testing*) dengan memanjangkan tempoh pengujian (*testing schedule).* Sebagai contoh, projek yang mempunyai 6 pusingan pengujian (*6 testing cycles*), boleh diselitkan pengujian regressi di 2 pusingan terakhir atau di pusingan terakhir. Ia bergantung kepada kecepatan SIT disempurnakan. Adakalanya, 6 pusingan pengujian yang diputuskan tidak mampu untuk menampung jumlah *test cases* disebabkan isu-isu yang berlaku (*showstopper*) ketika pengujian dijalankan. Ini akan menyebabkan pusingan istimewa (*special cycle*) dibuat atau ditambah mengikut budi bicara pengurus projek untuk melakukan pengujian regressi. Jika projek berada dalam keadaan *ahead schedule* atau *on schedule* maka pertimbangan untuk membuat keputusan lebih mudah. Jika *behind schedule*, pengurus projek sudah tentu berada dalam keadaan sukar untuk membuat keputusan. Jika pusingan istimewa dipersetujui untuk ditambah, maka peringkat pengujian SIT boleh berlarutan sehingga hampir dua bulan lamanya, jika dikira satu pusingan untuk satu minggu (*1 cycle for a week*). Tempoh satu pusingan untuk satu minggu ini adalah pusingan normal bagi pengujian untuk aplikasi mainframe kerana mengambil kira aktiviti *online input*

selama 1 atau 2 hari dan *batch run* selama 2 atau 3 hari dengan menganggap tiada isu kritikal berlaku yang boleh menyebabkan pengujian dihentikan (*On-hold status).* Namun begitu, satu pusingan untuk aplikasi lain sudah tentu berbeza.

Bayangkan, untuk satu projek berstatus sederhana besar, menggunakan metodologi *waterfall*, boleh memakan masa setengah tahun untuk disudahkan. Pernah berlaku di satu syarikat gergasi dunia yang beroperasi di Malaysia, satu projek berstatus besar memakan masa 2 tahun untuk disiapkan. Itupun dengan pelbagai isu yang tidak ada penyelesaian yang muktamad. Dalam tempoh 2 tahun itu, pengurus projek sudah bersilih ganti bertukar. Begitu juga dengan pembangun perisian dan penguji perisian. Sudah tentu projek yang begitu lama untuk disudahkan ini menelan belanja yang amat tinggi dan berkemungkinan boleh juga memberi kerugian jika terdapat isu selepas pelaksanaan perisian (*production issue).*

Ada beberapa isu lain dalam metodologi *waterfall* yang menunjukkan ia semakin tidak relevan untuk digunakan seperti yang disebut oleh Chris Greenough dan David Worth (2018) dalam slaid perkongsian mereka hasil kajian bertahun-tahun di CSED. Antaranya ialah:

1. Kelemahan utama *waterfall* ialah kesulitan menangani perubahan (*software requirement change*) selepas projek pembangunan sudah pun berjalan.
2. Setiap fasa mesti disiapkan sebelum berpindah ke fasa pembangunan seterusnya.
3. Pembahagian projek yang tidak anjal ke peringkat yang berbeza menjadikan ia sukar untuk memberi respon terhadap perubahan keperluan pelanggan.
4. Metod ini lebih sesuai jika keperluan setiap projek sudah difahami dengan sangat baik oleh setiap pemegang kepentingan (*stakeholder*) dan perubahan-perubahan yang hendak dilakukan adalah terhad semasa proses rekabentuk perisian.
5. Metod ini lebih sesuai digunakan untuk projek kejuruteraan sistem mega dimana sistem dibangunkan di beberapa laman.

Ulasan Ringkas Mengenai Pengujian Perisian Regressi Dalam Agile Scrum

Amir Ghahrai (2018) dalam artikelnya '*How to Overcame Agile Testing Challenges*' berkata:

"Dalam agile, pengujian bukan fasa/peringkat, ia adalah aktiviti. Pengujian bermula dari awal bahkan sebelum pembangunan (pengkodan) bermula."

Seperti aktiviti-aktiviti lain dalam agile, pengujian dilakukan secara berterusan. Tiada lagi keadaan dimana *testing team* bekerja di satu sudut lain bersama *testers* sahaja menanti *dev team* untuk menyerahkan secebis dari sistem untuk diuji. Begitu juga *dev team* tidak lagi bekerja di satu sudut berbeza yang lain. Dalam agile, semua bekerja dalam satu kumpulan yang besar. Ia akan bermula dengan sesi *grooming* setelah dokumen berkaitan projek diperolehi. Semua pihak harus bersama-sama dalam bekerja untuk membangunkan perisian. Sesi *grooming* selalunya sukar.

Mengenali dan memahami projek perisian yang hendak dibangunkan memerlukan kerjasama tinggi. Para penguji perisian perlu bekerjasama dengan *product owner* untuk mempelajari maklumat terperinci mengenai *user stories* yang disediakan. Kebiasaannya, *user stories* yang disediakan oleh *product owner* tidak mendalam dan memerlukan imaginasi yang tinggi untuk difahami. Sesi *walkthrough* mungkin perlu dilakukan berulang kali sekerap yang mungkin semasa *acceptance criteria* direka bagi memperjelas *user stories* tersebut. Contoh *user stories* yang di ambil dari www.agilemodeling.com adalah seperti di bawah:

- Students can purchase monthly parking passes online.
- Parking passes can be paid via credit cards.
- Parking passes can be paid via PayPal.
- Professors can input student marks.
- Students can obtain their current seminar schedule.
- Students can order official transcripts.
- Students can only enroll in seminars for which they have prerequisites.
- Transcripts will be available online via a standard browser.

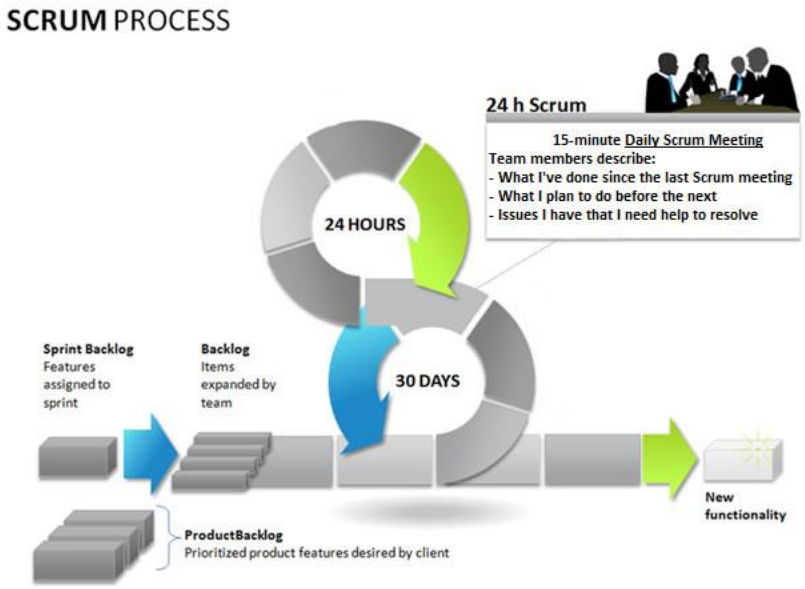
Kemudian barulah penguji perisian boleh memulakan aktiviti merangka *test cases.* Secara asasnya, dalam menyiapkan *test cases,* penguji perisian boleh merangka senario peringkat tinggi (*high level scenarios*). Ia mungkin sedikit sukar untuk difahami jika berlaku keadaan di mana, penguji menyerahkan tugas menguji perisian kepada penguji lain kerana senario peringkat tinggi tidak mudah untuk difahami. Namun begitu, jika berlaku sebarang perubahan (*requirement change*) maksud atau konteks masih lagi sama. Ada keadaan di mana, senario-senario dalam *test cases* harus diperincikan lagi supaya menjadi peringkat rendah (*low level scenarios*). Ia bergantung kepada perbincangan dengan pengurus projek yang menyemak *test cases* yang telah siap dirangka.

Semasa proses merangka *test cases,* projek kebiasaannya sudah pun berjalan. Aktiviti pengujian juga seharusnya bermula. Proses pengkodan dan pengujian ini sentiasa berulang-ulang 24 jam sehari sehinggalah pelanggan berpuashati. Langkah ini dipanggil sebagai *Sprint*. Kekerapan proses *sprint* hanya boleh berlarutan sehingga sebulan sahaja seperti yang diterangkan oleh Mishkin Berteig (2014) dalam artikelnya "*21 Tips On Choosing a Sprint Length*". Dalam tempoh itu, akan berlaku banyak perubahan-perubahan bagi memastikan produk yang dihasilkan bukan sahaja berkualiti tetapi memberi kepuasan kepada semua pemegang kepentingan terutama pelanggan. Dalam *agile,* pelanggan boleh memantau secara langsung dari awal proses pembangunan perisian. Ia berbeza berbanding *waterfall* di mana hasil (*results*) setiap peringkat hanya boleh diketahui melalui dokumen-dokumen (*exit criteria*) yang dikongsi sebelum memulakan sesuatu peringkat baru. Ia tidak boleh diketahui secara langsung. Ini menunjukkan model *agile* lebih telus (*transparent*).

Dalam setiap *sprint,* akan berlaku banyak perbincangan pantas dan pendek. *Scrum Master* biasanya akan mengendalikan sesi mesyuarat berdiri (*standup meeting*). Berdasarkan pengalaman dalam industry, sesi *standup meeting* ini boleh berlaku 2 atau 3 kali sehari. Kebiasaanya satu sesi pada sebelah pagi, satu sesi sebelum makan tengahari dan sesi terakhir pada sebelah petang. *Standup meeting* akan berlaku secara pantas selama 15 minit sahaja. Setiap ahli kumpulan *agile (agile team)* mesti membentangkan status gerakkerja masing-masing seringkas dan sepadat yang mungkin. Status *complete, on-going* dan *pending* akan dicatatkan dan

diambil maklum oleh semua ahli dalam kumpulan *agile* sebagai panduan gerakkerja keesokan hari.

Rajah 3: Proses Agile SCRUM



Sumber: https://www.cprime.com

Aktiviti pengujian sudah tentu akan berlaku dengan kerap. Ia sangat memenatkan kerana perubahan-perubahan yang tidak dijangka sehingga saat-saat terakhir. Dilema yang akan berlaku

ialah bagaimana hendak melakukan pengujian regressi memandangkan masa yang suntuk dengan jumlah penguji perisian yang terhad. Oleh sebab itu, dalam *agile* aktiviti pengujian tidak boleh dilakukan secara manual sahaja. Ia perlu dilakukan secara auto menggunakan *automation tools* tertentu bagi mempercepatkan aktiviti pengujian seperti Selenium, TestComplete dan lain-lain. Penulis juga sempat menggunakan JIRA sebagai platfom untuk melaporkan *defect* yang amat menyokong model *agile.* Semasa syarikat masih menggunakan metodologi *waterfall,* penulis menggunakan HP ALM dan ia juga boleh dianggap platfom yang baik untuk melaporkan *defect.*

Walaupun dibekalkan dengan *automation tools,* pengujian regressi masih akan berdepan dengan kesukaran untuk dilakukan. Berdasarkan pengalaman, antara cabarannya ialah:

1. Kekangan masa untuk menguji semua fungsi sedia ada (*BAU functions*) yang akan memberi risiko dari segi kualiti jika dilakukan dengan gopoh.
2. Kurang tenaga kerja (limited resources) untuk menguji semua senario sedia. Ini memaksa penguji perisian yang ada dan boleh bekerja untuk melakukan aktiviti pengujian tanpa henti sehingga berhari-hari lamanya. Berehat selama satu jam sehari adalah perkara biasa jika situasi ini berlaku.
3. Berdepan masalah untuk memilih senario sedia ada yang mana yang lebih utama dan penting kerana kebiasaannya semua fungsi BAU (*BAU Function*s) adalah penting. Tempoh bagi setiap sprint yang pendek memberi lebih tekanan untuk pengujian regressi dilaksanakan. Dalam masa yang sama, senario untuk produk baru harus disiapkan juga.

Penguji perisian dalam model *agile* acap kali berdepan dengan isu ini namun seperti yang diketahui umum, dalam model ini semua pemegang kepentingan termasuk pembangun dan penguji perisian harus memikirkan jalan penyelesaiannya. Ia membenarkan lagi kata-kata Andrian Cho (2012) dalam artikelnya "*Don't Just Do Agile. Be Agile",* ahli kumpulan harus bertanya kepada diri mereka sendiri dalam sesi *Sprint Retrospective* 3 soalan ringkas: "*what went well, what didn't go well, what concrete actions can we take to improve?*" Dalam sesi itu perbincangan harus telus. Ia seperti mesyuarat *post-morterm* yang mesti dibuat selepas habis

satu-satu program untuk menilai dan mendokumenkan segala aktiviti yang berlaku sepanjang program supaya ia boleh dijadikan rujukan pada masa depan.

Kesimpulan

Perbincangan mengenai pengujian perisian khususnya pengujian regressi dalam *waterfall* dan *agile* dalam aplikasi *mainframe* telah dikupas secara asas di dalam artikel ini. Segala maklumat yang dibentangkan boleh dikembangkan dan diperincikan terutama dari segi aspek teknikal dan proses dalam kedua-dua model. Kebanyakan isu yang dikupas di dalam artikel ini adalah menunjukkan apa yang berlaku secara realitinya dalam industri. Kelemahan yang berlaku dalam model *agile* bukan kerana model itu tetapi kerana ketidaksediaan pemegang kepentingan untuk beralih arah dari model *waterfall* yang lebih santai dan tidak menguntungkan bagi kebanyakan syarikat pembangunan perisian. Pekerja-pekerja senior dalam sektor ini mengalami kesukaran yang kritikal untuk menyesuaikan diri dengan model *agile* yang bukan sahaja mementingkan kualiti bahkan produktiviti dan kecekapan.

Rujukan

Winston W. Royce (1970). "Managing the Development of Large Software Systems" in: *Technical Papers of Western Electronic Show and Convention* (WesCon) August 25–28, 1970, Los Angeles, USA.

T.E. Bell & T.A Thayer (1976). “Software requirements: Are they really a problem?” in ICSE '76 Proceedings of the 2nd international conference on Software engineering. Pages 61-68

Wilfred Van Casteren (2017) “The Waterfall Model and Agile Methodologies : A comparison by project characteristics

Ed Scannell (2017), August 21). **“**The mainframe environment: A thing of the past or the future?”. Retrieved 20 December 2018 from https://searchdatacenter.techtarget.com

Mainframe Testing - Complete Tutorial. Retrieved December 1, 2018 from http://www.guru99.com

Maarten Winkels (2010). “Regression Testing with an Agile Mindset”. Retrieved December 20, 2018 from https://xebia.com/blog

User Stories : An Agile Introduction. Retrieved December 30, 2012 from www.agilemodeling.com

Amir Gharai (2018) "How to Overcome Agile Testing Challenges". Retrieved December 30, 2018 from https://www.testingchallenge.com

Mishkin Berteig (2014) "21 Tips on Choosing a Sprint Length". Retrieved December 31, 2018 from www.agileadvice.com

Andrian Cho (2012) "Do Just Do Agile. Be Agile". Retrieved December 31, 2018 from https://www.ibm.com