BAB VI
KESIMPULAN DAN SARAN

6.1 Kesimpulan

Penelitian ini telah berhasil mengidentifikasi risiko-risiko yang terjadi di perusahaan ketika bekerja menggunakan Scrum dalam mengembangkan produk perangkat lunak. Pada penelitian ini, penulis berhasil menemukan 17 risiko dari hasil mewawancarai 3 perusahaan yang menggunakan Scrum. Ketiga perusahaan tersebut adalah perusahaan Kurio, Tokopedia, dan HappyFresh. Penulis juga telah mengelompokan risiko-risiko tersebut kedalam 4 kategori yakni kategori manajemen proyek, kategori organisasi, kategori teknis, dan kategori eksternal. Pengkategorian tersebut didasarkan pada Risk Breakdown Structure yang ada pada buku A Guide to The Project Management Body Of Knowledge oleh (Project Management Institute, 2008).

Risikopada kategori manajemen proyek merupakan risiko-risiko yang terkait dengan bagaimana proses-proses yang terjadi dalam menjalankan proyek. Risiko-risiko yang terkait dengan acara pada Scrum juga termasuk kedalam risiko kategori manajemen proyek seperti Sprint Planning kurang matang, Daily Scrum kurang efektif, Sprint Retrospektive tidak rutin diadakan, dan tambahan pekerjaan ditengah Sprint. Risiko terkait dengan acara Scrum termasuk kedalam kategori manajemen proyek karena Scrum itu sendiri merupakan upaya untuk manajemen suatu pengerjaan produk perangkat lunak. Risiko-risiko pada kategori manajemen proyek ini disebabkan oleh banyak hal seperti karakter

atau pengalaman kerja dari tim Scrum, dan peranan manajemen dalam Scrum.Risiko pada kategori manajemen proyek secara umum mengakibatkan proses pengembangan perangkat lunak menjadi kurang baik.

Risiko pada kategori organisasi merupakan risiko-risiko yang terkait dengan sumber daya di organisasi. Sumber daya tersebut salah satunya adalah sumber daya manusia. Risiko yang termasuk kedalam kategori organisasi ini antara lain adanya *dependency,* bekerja dengan tidak maksimal, jumlah atau komposisi anggota tim tidak efektif, tidak adanya Product Owner yang khusus, kurangnya keterlibatan QA didalam proses Scrum, dan *meeting* tambahan yang mengganggu. Risiko-risiko ini secara umum disebabkan karena kurang tepatnyacara organisasi mengatur sumber daya manusia yang ada untuk bekerja dengan baik dalam mengembangkan produk perangkat lunak. Risiko pada kategori organisasi secara umum menyebabkan kurang baiknya proses interaksi dalam pengembangan perangkat lunak sehingga kualitas produk yang dihasilkan menjadi buruk.

Risiko pada kategori teknis terkait dengan risiko-risiko dalam melakukan pemrograman atau *coding*, dokumentasi, dan kebutuhan. Risiko yang termasuk kedalam kategori teknis adalah kebutuhan yang tidak jelas, tujuan proyek tidak jelas, kurangnya dokumentasi dan ketidakcocokan *pair programming*.Risiko pada kategori ini biasanya terjadi karena sifat atau kualifikasi dari anggota tim Scrum yang kurang baik. Akibatnya, risiko yang berada pada kategori ini menyebabkan hasil dari produk perangkat lunak yang

dibangun memiliki kualitas yang kurang baik dan proses pengembangan perangkat lunak terhambat.

Risiko pada kategori eksternal merupakan kategori untuk risiko yang berasal dari luar proyek. Penelitian ini hanya menemukan satu risiko eksternal yakni perubahan kebutuhan yang sangat cepat. Risiko ini disebabkan oleh *client* atau *market* yang merubah kebutuhan dengan cepat dan harus diterima oleh organisasi atau perusahaan yang memiliki *mindset* Agile. Akibatnya, waktu pengerjaan proyek akan semakin lama karena ada banyak perubahan dan proses pengembangan perangkat lunak tidak dapat dijalankan dengan baik.

6.2 Saran

Pada penelitian ini telah diidentifikasikan risiko-risiko yang terjadi pada beberapa perusahaan yang mengimplementasikan Scrum. Risiko-risiko ini merupakan risiko-risiko yang terjadi didalam penggunaan kerangka kerja Scrum. Risiko yang terjadi ini dapat menghambat tim Scrum dalam menggunakan kerangka kerja Scrum dengan baik sehingga dapat membuat proses pengembangan perangkat lunak menjadi terganggu. Risiko-risiko tersebut juga sudah dikelompokan kedalam beberapa kategori untuk memudahkan perusahaan dalam pengambilan keputusan terkait dengan risiko yang ada.

Masing-masing risiko yang telah teridentifikasi tersebut dapat dikembangkan lebih mendalam lagi kedalam penelitian yang baru. Penulis menyarankan penelitian untuk memperdalam masing-masing risiko yang teridentifikasi untuk mengetahui seberapa besar pengaruh risiko tersebut terhadap organisasi atau

proyek. Penelitian selanjutnya juga dapat melakukan analisis kualitatif risiko. Untuk itu, diperlukan datamengenai frekuensi terjadinya risiko. Frekuensi terjadinya risiko dan dampak yang dihasilkan dapat menghasilkan *risk rangking* untuk mengetahui risiko mana yang seharusnya ditangani terlebih dahulu.*Risk ranking* tersebut dapat dicapai dengan membangun *probability impact matrix.* Untuk itu, perlu adanya data frekuensi terjadi risiko beserta dengan ukuran dari dampak risiko agar dapat membangun *probability impact matrix* yang dapat menunjukan risiko yang harus ditangani oleh perusahaan terlebih dahulu.

6.3 Implikasi Manajerial

Penulis telah memaparkan hasil penelitian berdasarkan data yang diperoleh. Didalam penelitian ini, penulis telah menghasilkan risiko-risiko yang muncul dari penggunaan kerangka kerja Scrum dalam pengembangan perangkat lunak. Risiko-risiko tersebut telah dijabarkan beserta dengan penyebab terjadi risiko dan akibat yang ditimbulkan apabila risiko tersebut terjadi.

Hasil dari penelitian ini dapat digunakan oleh manajer untuk membantu pengambilan kebijakan yang dapat membantu organisasi agar dapat bekerja dengan lebih baik dalam mengembangkan perangkat lunak. Berdasarkan hasil penelitian ini, manajer dapat melakukan hal-hal berikut:

- Manajer dapat mengambil kebijakan terkait bagaimana cara mencegah terjadinya risiko dan mengatasinya apabila terjadi sesuai dengan risiko yang telah

teridentifikasi dan dideskripsikan pada penelitian ini. Risiko tidak dapat dihilangkan, tetapi dapat dikurangi atau ditransfer. Manajer perlu menentukan kebijakan yang tepat terkait penanganan risiko yang ada.

- Manajer dapat menentukan risiko yang harus ditangani terlebih dahulu berdasarkan dampak dari risiko yang ditemukan. Setiap perusahaan pasti memiliki kriteria masing-masing mengenai masalah yang harus ditangani terlebih dahulu. Pada umumnya perusahaan pasti mengatasi risiko yang dapat menghambat bisnis dari perusahaan tersebut. Dengan mengetahui risiko mana saja yang berakibat pada bisnis perusahaan, manajer dapat memberikan perhatian lebih pada risiko tersebut. Klasifikasi risiko pada penelitian ini membantu manajer agar dapat mengetahui area mana yang harus diberi perhatian lebih.

DAFTAR PUSTAKA

Akif, R. & Majeed, H., 2012. Issues and Challenges in Scrum Implementation. *International Journal of Scientific & Engineering Research,* 3(8), pp. 1-4.

Anand, R. V. & Dinakaran, M., 2015. Issues in Scrum Agile Development Principles and Practices in Software Development. *Indian Journal of Science and Technology,* 8(35), pp. 1-5.

Barker, T. T., 2003. Documentation for Software and IS Development. *Encyclopedia of Information Systems,* Volume 1, pp. 683-693.

Bassil, Y., 2012. A Simulation Model for the Waterfall Software Development Life Cycle. *International Journal of Engineering & Technology (IJET),* 2(5).

Bloch, M., Blumberg, S. & Laartz, J., 2012. *Delivering large-scale IT projects on time, on budget, and on value,* s.l.: McKinsey&Company.

Burnard, P. et al., 2008. Analysing and Presenting Qualitative Data. *British Dental Journal,* 204(8), pp. 429-432.

Cambridge University Press, 2016. *Cambridge Dictionary.* [Online]
Available at: http://dictionary.cambridge.org/dictionary/english/introvert
[Accessed 15 September 2016].

Cooper, D. F., Grey, S. & Raymond, G., 2005. *Project Risk Management Guidelines: Managing Risk in Large Project and Complex Procurement.* Chichester: John Wiley & Sons Ltd.

Deloitte, 2015. *Deloitte's 9th Global Financial Services Risk Management Survey.* [Online] Available at: http://www2.deloitte.com/global/en/pages/about-deloitte/articles/financial-services-risk-management-survey-press-release.html#

Dey, P. K., Kinch, J. & Ogunlana, S. O., 2007. Managing Risk in Software Development Projects: A Case Study. *International Journal of Risk Assesment and Management,* 107(2), pp. 284 - 303.

Eler, M. M., Endo, A. T. & Durelli, V. H., 2016. An empirical study to quantify the characteristics of Java programs that may influence symbolic execution from a unit testing perspective. *The Journal of Systems and Software,* 121(20), p. 281–297.

Febriayanti, A. & Hidayanto, B. C., 2012. Manajemen Resiko Pada Pengelolaan Data di Bagian Pengelolaan Data PT. Petrokimi Gresik. *Jurnal Teknik POMITS,* 1(1), pp. 1 - 6.

Flick, U., 2009. *An Introduction to Qualitative Research.* 4th ed. Reinbek bei Hamburg: SAGE.

Gandomani, T. J. & Nafchi, M. Z., 2016. Agile Transition and Adoption Human-Related Challenge and Issue: A Grounded Theory Approach. *Computer in Human Behavior,* Volume 62, pp. 257-266.

Gregory, P. et al., 2016. The Challenges That Challenge:Engaging With Agile Practitioner' Concerns. *Information and Software Technology.*

Hillson, D., 2003. Using a Risk Breakdown Structure in Project Management. *Journal of Facilities Management,* 2(1), pp. 85-97.

Holzmann, V. & Spiegler, I., 2010. Developing Risk Breakdown Structure for Information Technology Organization. *International Journal of Project Management,* 29(5), pp. 537-546.

Iskandar, I., 2011. Manajemen Risiko Teknologi Informasi Perusahaan Menggunakan Framework RiskIT (Studi Kasus: Pembobolan PT. Bank Permata, Tbk. *Jurnal Sains, Teknologi dan Industri,* 9(1), pp. 104 - 115.

Lindsjorn, Y. et al., 2016. Teamwork Quality and Project Success in Software Development: A Survey of Agile Development Teams. *The Journal of Systems & Software.*

Mahalakshmi, M. & Sundararajan, D. M., 2013. Traditional SDLC Vs Scrum Methodology – A Comparative Study. *International Journal of Emerging Technology and Advanced Engineering,* 3(6), pp. 192 - 196.

Martin, R. C., 2012. *Agile Software Development.* 1 ed. Upper Saddle River: Alan R Apt.

Partogi, J., 2015. *Manajemen Modern dengan Scrum.* 1 ed. Yogyakarta: ANDI.

Paul, S. & Singh, K. J., 2012. Be Agile: Project Development With Scrum Framework. *Journal of Theoretical and Applied Information Technology,* 40(1), pp. 105-112.

Project Management Institute, 2008. *A Guide to The Project Management Body Of Knowlege.* 4th ed. Pennsylvania: Project Management Institute.

Ruler, B. v., 2015. Agile Public Relation Planning: Reflective Communication Scrum. *Public Relations Review,* 41(2), pp. 187 - 194.

Schwaber, K. & Sutherland, J., 2013. *The Scrum Guide.* [Online]
Available at: http://www.scrumguides.org/scrum-guide.html
[Accessed 5 September 2016].

Schwalbe, K., 2011. *Information Technology Project Management.* Boston: Course Technology.

Shrivasava, S. V. & Rathod, U., 2015. Categorization of Risk Factors for Distributed Agile Projects. *Information and Software Technology,* 58(1), pp. 373 - 387.

Sohi, A. J., Hertogh, M., Rekveldt, M. B. & Blom, R., 2015. *Does Lean & Agile Project Management Help Coping With Project Complexity.* Panama, Elsevier.

the Agile Manifesto, 2001. *Manifesto for Agile Software Development.* [Online]
Available at: http://agilemanifesto.org/iso/id/manifesto.html
[Accessed 9 September 2016].

Tomanek, M. & Juricek, J., 2015. Project Risk Management Model Based on Prince2 and Scrum Framework. *International Journal of Software Engineering & Applications (IJSEA),* 6(1), pp. 81 - 88.

Versionone, 2011. *State of Agile Survey,* Atlanta: Versionone.com.

Yin, R. K., 2011. *Qualitative Research from Start to Finish.* 1st ed. New York: A Division of Guilford Publication, Inc..

LAMPIRAN I

PROFIL PERUSAHAAN

Dalam penelitian ini, peneliti mencoba untuk menentukan objek penelitian. Peneliti berhasil mendapatkan persetujuan dari 3 perusahaan yang mau menjadi sumber data bagi penelitian ini. Berikut merupakan perusahaan yang menjadi sumber data penelitian ini:

No	Perusahaan	Deskripsi Singkat	Alamat
1	Kurio	Kurio merupakan perusahaan yang membuat produk aplikasi penyedia layanan berita yang cerdas yakni aplikasi Kurio. Kurio merupakan anak perusahaan dari MCM (Merah Cipta Media). Kurio membantu menemukan konten berita yang pengguna inginkan.	Wisma 77 Tower 2 Lantai 3, Jalan Letjen. S. Parman No. 77, Daerah Khusus Ibukota Jakarta
2	Tokopedia	Tokopedia merupakan perusahaan internet yang mengizinkan individu dan pemilik bisnis di Indonesia membuka dan mengatur toko online mereka sendiri dengan mudah dan gratis. Tokopedia menyediakan pengalaman penjualan online yang lebih baik pada penjual sehingga penjual dapat memberikan pengalaman belanja yang lebih baik kepada pembelinya.	Wisma 77 Tower 2 Lantai 2, Jl. Letnan Jenderal S. Parman Kav. 77, Slipi, Palmerah, Jakarta Barat, Daerah Khusus Ibukota Jakarta
3	HappyFresh	HappyFresh adalah *grocery online* yang pertama dan tercepat pertumbuhannya di asia tenggara. Bersama dengan semua tim yang berbasis di Jakarta, HappyFresh telah dengan cepat dikembangkan ke Malaysia, Indonesia dan Thailand dengan rencana pertumbuhan yang	16th Floor, Talavera Office Park, Jl. Letjend TB Simatupang, Kav. 22 – 26, Cilandak, Jakarta Selatan 12430

		ambisisius.	

Sumber: kurio.co.id, tokopedia.com/about, HappyFresh.com

LAMPIRAN II

DATA NARASUMBER

Kode	Nama	Jabatan	Scrum Role	Perusahaan
K1	Nuruddin Ashr	Engineering Manager	Product Owner	Kurio
K2	Hendy Christianto	Mobile Lead	Development Team	Kurio
K3	Steven Tiole	Software Engineer	Development Team	Kurio
K4	Sella	UI Designer	Development Team	Kurio
K5	Arie	Senior API Engineer	Development Team	Kurio
T1	Christian Antonius	Quality Assurance	Development Team	Tokopedia
T2	Kenneth Vincent	IOS Pengembang	Development Team	Tokopedia
T3	Tri Nugraha	Product Owner	Product Owner	Tokopedia
T4	Hafizh Herdi	Android Pengembang	Development Team	Tokopedia
T5	Ryan Handy	UX Designer	Development Team	Tokopedia
T6	Patric Alexander	Software Engineer	Scrum Master	Tokopedia
T7	Ryandy Law	Web Pengembang	Development Team	Tokopedia
H1	Nu'man Naufal	Junior Software Engineer	Development Team	HappyFresh
H2	Rizky Maulana	Backend Engineer	Development Team	HappyFresh
H3	Artanto Ishaam	Project Manager	Scrum Master	HappyFresh
H4	Dedi Setiadi	Quality Assurance	Development Team	HappyFresh
H5	Isnan Freauseda	Technical Lead	Development Team	HappyFresh
H6	Rheza Sastra	IOS Pengembang	Development Team	HappyFresh
H7	Hanifa Azhar	Product Manager	Product Owner	HappyFresh
H8	Ivan Afandi	Lead UI/UX Designer	Development Team	HappyFresh

LAMPIRAN III

PERTANYAAN WAWANCARA

Bagian I - Identidas Narasumber dan Waktu Wawancara

1. Jadwal Wawancara

a. Tanggal / Hari :

2. Identitas Narasumber
    a. Jenis Kelamin :
    b. Usia :
    c. Jabatan :
    d. Scrum Role :

Bagian II - Pertanyaan Semi Terstruktur

a. Pertanyaan wawancara untuk pengembang

1. Apakah risiko dalam penggunaan Scrum sebagai metodologi pengembangan perangkat lunak?
2. Bagaimana cara mengatasi risiko yang terjadi dalam pengembangan?
3. Berapa kali melakukan *meeting* selama satu minggu? Apakah pernah merasa terganggu untuk melakukan *meeting* yang diluar Scrum *meeting*?
4. Apakah Sprint Retrospective selalu dilakukan di setiap akhir Sprint? Apakah berjalan dengan baik? Bagaimana jika Sprint Retrospective tidak dijalankan?
5. Apakah sering terjadi masalah pribadi dalam tim Scrum? Apabila ada, bagaimana cara anda dalam mengatasi masalah pribadi tersebut?

b. Pertanyaan wawancara Product Owner

1. Apakah risiko dalam penggunaan Scrum sebagai metodologi pengembangan perangkat lunak?
2. Bagaimana cara mengatasi risiko yang terjadi dalam pengembangan?
3. Siapa yang menentukan prioritas Product Backlog Item di Perusahaan anda? Bagaimana cara anda menentukan prioritas Product Backlog Item mana yang harus dikerjakan terlebih dahulu?

c. Pertanyaan wawancara Scrum Master

1. Apakah risiko dalam penggunaan Scrum sebagai metodologi pengembangan perangkat lunak?

2. Bagaimana cara mengatasi risiko yang terjadi dalam pengembangan?
3. Jabarkan secara singkat perbedaan implementasi Scrum di perusahaan ini dengan Scrum secara umum!
4. Apabila ada anggota baru pada tim, bagaimana cara melatih anggota baru tersebut?
5. Apakah pernah dalam 1 Sprint, tidak berhasil menyelesaikan semua User Story yang telah disepakati di awal? Bagaimana cara menangani Sprint yang User Story nya tidak dapat diselesaikan oleh pengembang dalam 1 Sprint?

Bagian III – Pertanyaan Semi Terstruktur (Theory-Driven)

1. Bagaimana pendapat anda mengenai tujuan proyek yang kurang jelas?
2. Bagaimana pendapat anda mengenai *requirement* yang tidak jelas?
3. Bagaimana pendapat anda mengenai konflik *requirement* antar Product Owner?
4. Bagaimana pendapat anda mengenai prioritas *requirement* yang tidak memadai?
5. Bagaimana pendapat anda mengenai perubahan *requirement* yang sering terjadi?
6. Bagaimana pendapat anda mengenai pelaksanaan Technical Debt di dalam tim?
7. Apakah anda merasa ada masalah ketidakcocokan dalam Pair Programming?
8. Apakah terdapat dokumen yang memadai untuk *testing*?
9. Bagaimana keefektifan *standupmeeting* disini?
10. Apakah terdapat perbedaan proses Scrum antar tim?
11. Apa*definition of done* yang ada di perusahaan ini?
12. Bagaimana pengaruh *velocity* didalam proses Scrum?
13. Bagaimana pengaruh jumlah anggota tim didalam proses Scrum dengan proses *development*?
14. Bagaimana peranan Business Analyst didalam Scrum?
15. Bagaimana komunikasi antar tim yang terjadi?
16. Bagaimana *skill* komunikasi yang dimiliki tim Scrum?
17. Apakah terdapat dokumentasi produk akhir?
18. Bagaimana koordinasi antar tim yang terjadi di perusahaan ini?

19. Bagaimana kolaborasi yang terjadi antara pengembang dan QA?
20. Bagaimana anda menilai kehandalan kinerja PO?

LAMPIRAN IV

TRANSKRIP WAWANCARA DAN OPEN CODING

Transkrip wawancara yang penulis lampirkan disini berasal dari rekaman hasil wawancara dengan 3 perusahaan yakni Kurio, Tokopedia, dan HappyFresh. Pada transkrip wawancara ini, penulis telah menggabungkannya dengan *open coding* yang merupakan bagian dari teknik analisis data. Pada setiap pernyataan dari narasumber, penulis telah memberikan tanda "coding" yang merupakan bagian untuk mengisikan kode yang ditemukan dari pernyataan narasumber. Setiap pernyataan narasumber tersebut penulis masukan kedalam kotak tabel agar dapat mudah dibedakan antara satu jawaban pernyataan dengan yang lainnya.

Pada transkrip wawancara ini,penulis memberikan kode pada kolom nomor untuk mempermudah penulis dalam menelusuri pernyataan wawancara narasumber dari kode yang telah dikumpulkan pada lampiran. Pengkodean pada kolom nomor dilakukan dengan aturan sebagai berikut:

" XA.B "

X = Kode Perusahaan (K untuk Kurio, T untuk Tokopedia, H untuk HappyFresh).

A = Urutan Narasumber yang diwawancarai di perusahaan tersebut.

B = Urutan pernyataan jawaban dari narasumber yang bersangkutan.

Pada transkrip wawancara ini, penulis memberikan kode N untuk dialog narasumber. Karena wawancara dilakukan oleh 2 orang pewawancara, maka diberikan kode T untuk Toni Indrawan dan kode S untuk Sari Aritonang.

TRANSKRIP WAWANCARA SKRIPSI
PT Kurio

No	**Nama: Nuruddin Ashr** **Jabatan: Engineering Manager** **Scrum Role: Product Owner / Scrum Master**
K1.1	T: Sebagai PO, Apa risiko penggunaan Scrum? N: Risiko nya dari sisi *product* adalah masalah 1. Masalah *delivery*. Kalau di konvensional, kalau sudah buat *deadline*, *engineer* mau ga mau selesaikan sesuai *deadline*. *Sometimes*, kita kayak ga mau tahu pokoknya tanggal sekian sudah harus beres. Coding: *Deadlineengineer/developer*
K1.2	T: Jadi *engineer*nya dipaksa untuk selesai sesuai *deadline*. Ada risiko lain? N: Risiko lain ada, Cuma ini risiko yang langsung kelihatan oleh PO. Misalkan dari yang saya alami, kalau kita ga pakai Scrum, *spendtimeproject* 1 bulan hingga 2 bulan, jadi lebih panjang. Ini bergantung dari *style* masing-masing. Tapi terkadang, kita melihat hasil saat *project* sudah beres. Kadang tidak sesuai ekspektasi saja. Kalau tanpa Scrum. Oh *sorry*, ini risikonya Scrum. Coding: Style penggunaan Scrum masing-masing berbeda.
K1.3	T: Iya, Berarti risikonya, Cuma masalah *delivery* yang memaksa Pengembang selesai sesuai *deadline*. N: Kalau tanpa Scrum, kita ga bisa menentukan estimasi proyek dapat selesai kapan karena kita cuma dapat melihat 1 Sprint. Paling saat sudah berjalan 2 atau 3 Sprint baru ketahuan kapan akan selesai. Tetapi terkadang dalam satu Sprint tidak selalu steril. Ada saja kerjaan lain yang diluar Sprint yang masuk dan akibatnya, satu Sprint itu kayak variable. Jadi ga selalu sama. Jadi kita punya estimasi, tetapi tidak dapat dipastikan. Sifatnya estimasi tetapi bukan *deadline*. Jadi cuma bisa diprediksi saja. Kita ga bisa pastikan dalam 3 bulan kita bakalan punya *feature* seperti ini. Coding: Proyek tidak dapat diestimasi di awal Sprint.
K1.4	T: Iya, karena tergantung dari *velocity* tim Scrumnya sendiri. N: Betul.

	Coding: -
K1.5	S: Terus kalau untuk mengatasi *engineer* yang tidak mau tahu kalau *deadline* nya segini, gimana? T: Ada upaya pencegahannya ga? Atau cara mengatasi agar *engineer* dapat tetap *commit* sesuai *deadline*. N: Sebenar nya gini sih, kalau mau soal *deadline* ya, ini kita kalau di Scrum, hal yang coba kita tackle sih kita bikin *shortdeadline*. Ini yang bisa kita tekankan. Karena pakainya Scrum, contoh ya, kita kemarin punya *velocity* 40. Tetapi ditengah jalan ada kerjaan baru masuk. Karena ada kerjaan baru yang *priority*-nya lebih tinggi. Saat yang 40 itu sudah ga *commit* lagi, kita agak sulit menentukan *velocity* yang baru karena sudah terganggu di tengah jalan. Jadi, tapi kalau ga ada hal seperti itu, kita akan lebih mudah untuk *please* dong *commit*. Dalam 2 minggu selesaikan deh. Coding: *Shortdeadline*, perubahan *velocity* di tengah Sprint, komitmen *developer* dalam Sprint
K1.6	T: Oh, berarti ini lebih menekankan pemikiran *engineer*nya sendiri supaya *commit*. N: Sama seperti *deadline* yang panjang, Cuma ini *deadline* nya diperkecil jadi 2 minggu atau 1 minggu. Hanya saja, kalau di Scrum, kan sifatnya adaptif. Ketika buat *spend* waktu 2 minggu, ada saja kerjaan masuk. Dan kalau itu terjadi, komitmen untuk 40 nya sudah berubah. Kalau misalkan ga ada, kita selalu *stick*, *pleasestick* pada 40 itu. Coding: Adaptif (kerjaan lain bisa masuk ditengah Sprint), komitmen *developer* dapat berubah
K1.7	S: Biasanya, prioritas Product Backlog Item itu, nentuinnya gimana sih pak? N: Kalau PBI sih dari sisi *product*nya. Kalau dari kita, kita punya *roadmap* mau tambahin fitur apa saja. Kadang-kadang prioritas itu berdasarkan keinginan. Kayaknya yang ini lebih bagus, lebih dulu. Atau misalkan kita ingin fitur yang keren muncul lebih dulu. Kalau di kita, kita identifikasikan berdasarkan KPI nya. Key Performance Indicator. Kita punya *metric* apa yang mau kita raih. Contoh, kita pengen semakin banyak orang dapat buka artikel. Nah, fitur apa yang kita butuhkan untuk membuat hal itu terjadi. Coding: Prioritas PBI berdasarkan keinginan,

	penentuan Product Backlog berdasarkan KPI
K1.8	T: dan yang menentukan PBI itu biasanya? N: PO Coding: PO menentukan PBI.
	T: Berikutnya, kita sebenarnya sudah punya list risiko untuk penggunaan Agile yang kami peroleh dari jurnal di India. Hanya saja, kami hendak mengkonfirmasi risiko itu apakah sesuai dengan konteks di Indonesia.
K1.9	T: Yang pertama adalah tujuan proyek yang kurang jelas. Kira-kira selama bapak menggunakan Scrum di Kurio ini, pernah ga terjadi tujuan proyek yang kurang jelas. N: Selama ini cukup jelas. Karena kita tidak terlalu besar. Biasanya kita selalu kasih tahu tujuan kita apa. Coding: Tujuan proyek cukup jelas, Proyek yang tidak terlalu besar membuat tujuan lebih jelas.
K1.10	T: Berarti selama ini selalu jelas-jelas saja karena *goal*nya selalu dikasih tahu. N: iya Coding: Tujuan proyek jelas.
K1.11	T: Kemudian, kalau *requirement* tidak jelas bagi tim? N: Requirement sih cukup jelas. Kalau di Scrum, kadang-kadang *requirement* tidak 100% baku. Kalau Scrum, yang di-*encourage* itu *corporation*-nya. Jadi kalau ada *requirement* yang belum jelas, biasanya langsung di-*explore* harus bagaimana *requirement*-nya. Karena kadang-kadang kita masuk estimasi pun masih banyak yang di-*refine* lagi. Banyak juga yang *interupt*, misalnya saat mau masuk ke *story* A harus ada *story* B dulu nih. Ketika kita jalan, memang *requirement*-nya tidak 100%. Ketika kita estimasi bisa muncul tambahan-tambahan lagi. Bahkan saat jalan, kita bisa identifikasi juga ternyata ada yang kurang. Coding: kebutuhan jelas, inisiatif meng-*explore* kebutuhan
K1.12	T: Berarti, pada saat awal umumnya tidak jelas. Dan frekuensi terjadinya dalam satu Sprint itu kecil sekali. N: Tidak 100%. Paling 70% jelas. Jadi kadang-kadang,

	si PO harus buat definisi *product* yang dia mau seperti apa. Tapi ga langsung ready. Biasanya kita langsung bahas di *planning*. 95% tidak terjadi. Coding: Kebutuhan kurang jelas di awal Sprint, PO membuat definisi produk yang jelas.
K1.13	T: Berarti 5 % nya terjadi. Nah, kalau 5% nya ini terjadi, bagaimana cara mengatasinya? Atau menggunakan kolaboratif tadi? N: Kalau misalkan kita bisa, dalam Scrum, 1 *cycle* ada sesuatu yang bisa kita punya, ada *value*-nya untuk *user*. Depends on the *user*. Kalau *user* nya pembaca berarti si pembaca. Fiturnya *curator*, ya berarti untuk *curator*. Nanti kalau dilihat *curator* bisa pakai ga? Kalau ga, kita perlu tambahin *story* itu biar *user* bisa pakai. Kita biasanya numpang ke *story* yang sudah ada. Biasanya kita *take a note*. Kalau pakai *sticky note*, kita tambahin tanda titik yang artinya *story* ini ga relevan. Atau kita tambahin *story* nya. Intinya kita tahu, secara hi*story*, kalau satu Sprint tim ga bisa *commit*, karena 1 atau 2 hal yang kita *unplan*, yang kita *missed* prediksi. Coding: Tandai *story* yang sudah tidak relevan.
K1.14	T: Nah, kalau misalkan *missed* prediksi itu terjadi, akibatnya apa ya? N: *Worstcase*-nya kita ga bisa *deliver* semua *story*. Tapi, walaupun itu terjadi, tetapi biasanya sih kita *story* yang *priority*-nya paling rendah yang tidak bisa kita *deliver*. Paling kita, kalau tinggal sedikit sekali, kita cari waktu tambahan untuk menyelesaikan itu. Tetapi kalau masih banyak, kita *carry over* to *next*Sprint. Coding: *Lowprioritystory* tidak di-*deliver*, *story* yang tidak selesai dilanjutkan ke Sprint selanjutnya.
K1.15	T: Nah kemudian, kita masuk ke risiko selanjutnya. Adanya konflik *requirement* antar Product Owner. Nah, ini sebenarnya *case* untuk jika seandainya ada lebih dari 1 PO di perusahaannya. N: Untuk disini sih, PO cuma 1, jadi tidak valid. Coding: Konflik kebutuhan tidak terjadi pada perusahaan dengan 1 PO.
K1.16	T: Prioritas *requirement* yang tidak memadai. Kesalahan memepriori*taskan* *requirement*. Kurang

	memadai ini maksudnya salah mengambil strategi, seharusnya yang ini yang dikerjakan dulu, eh pada implementasi ternyata harusnya ada yang lain yang dikerjakan terlebih dahulu. N: *Somehow* sih enggak ya. Kalau salah prioritas, sebenarnya dari semua yang kita lakukan, secara teknis salah prioritas ga ada. Contoh, salah prioritasnya *storydependency*. Tapi kalau salah prioritas, enggak. Karena kita punya KPI untuk dikejar, paling yang kita ambil ini ternyata tidak punya *impact* yang begitu bagus ke *user*. Tapi kita ga bilang itu sebagai suatu kesalahan. Tapi memang cara kerjanya begitu kalau di *startup*. Kalau mau *deliversomething*, kita coba dulu satu, kalau ga sesuai, kita coba lagi yang lain. Coding: *StoryDependency*, Salah prioritas *story*, *impact* untuk *user* kurang baik.
K1.17	T: Kemudian, perubahan *requirement*. Ini mungkin agak jelas ya. Perubahan *requirement*-nya sering ga terjadi? N: Perbuahan *requirement* sebenarnya ada. Kalau iya, ga begitu sering. *So far* disini, adanya perubahan detail. Kalau *requirement* tidak ada. Coding: Perubahan detail kebutuhan.
K1.18	T: Untuk perubahan detail itu sendiri, apa sih yang membuat detail itu harus berubah? N: Mostly sih kayak contoh ya, karena yang kita jalanin, kita ga mau bertele-tele. Kita punya UI. Kita sudah coba pikirin *flow*-nya seperti apa. Kadang-kadang ada yang *missed*. Kalau masuk ke implementasi, kan kita tahu gimana caranya jika kita ingin ngelakuin ini. Sedangkan ada hal-hal yang perlu kita perhatiin, yang belum pernah kita bahas sama sekali. Secara visual sudah oke, tapi kita ga bisa habisin waktu untuk berpikir ini *flow*-nya sudah oke belum sih karena sekilas kita berpikir sudah komplit. Oleh karena itu, saat implementasi ada perubahan detail atau penambahan. Yang kedua, kalau di-*app* biasanya masalah interaksi atau UI. *Sometimes*, kita merasa ini terlalu sulit bagi *user*, kita ganti saja *interface*-nya. Coding: Planning yang kurang detail, kesalahan *userexperience*.
K1.19	T: Kalau seandainya risiko tersebut terjadi, dampak bagi keseluruhan proyeknya bagaimana?

	N: Malah bagus kok. Contohnya begini, ketika kita mau achieve sesuatu yang lebih sulit, sebenarnya kalian bisa nawar, Kalau itu lebih sulit, atau ga sesuai sama standar. Contoh android dan iOS, kalau android ada *materialdesign*. Desainer buat dengan *style* mereka karena akan lebih gampang jika dibuat mengikuti material *design*. Biasanya, mereka *came up* dengan ide-ide, gimana bikin kayak gini, karena secara performa lebih cepat, dan lebih hemat waktu. *So far*, kita *fine* dengan itu. Coding: Perubahan detail kebutuhan berdampak positif.
K1.20	T: Dan ini frekuensi nya jarang? N: bisa sering. Coding: Sering terjadi perubahan detail kebutuhan.
K1.21	T: berapa persen kemungkinan terjadinya dalam 1 srint? N: 20 – 30 deh. Sebenarnya ini agak, bias. Biasanya kita punya 2 bagian *user*, ada bagian *curator*, ada bagian *user*. Untuk *user*, kita punya *design* tapi ga mirip persis. Untuk produk internal, kita punya lisan dan *wireframe*. Ketika sudah jadi, ga 100% seperti yang dibayangkan, jadi kita ga punya sesuatu yang *fix*. 30 % sih selalu ada, untuk yang kita punya *design*-nya. Untuk yang belum punya *design*-nya, kita selalu nego untuk detailnya. Coding: Detail desain dapat dinegosiasi.
K1.22	T: Kemudian ini, manajemen Technical Debt yang kurang. Kita ingin tahu manajemen Technical Debt di kurio gimana, dilakukannya pas kapan? Dan pasti dilakukan atau tidak? N: Kita ga suka sama Technical Debt. Kita selalu minta Technical Debt dimasukan ke *story*. Ini menyangkut sama *story* X. Maka lakukanlah Technical Debt di *story* X. Tapi jika *impact*-nya hampir semua aspek tersentuh, kita akan coba cari waktu yang lumayan senggang untuk melakukannya. Coding: *Technical Debt* masuk kedalam *story*.
K1.23	T: kalau dari Technical Debt itu sendiri biasanya dilakukannya pas kapan? N: Diluar Sprint. Kita kemarin *mobile* tim, *bugfix*. Dan banyak pekerjaan di *backend*. Pada masa-masa

	seperti itu dimasukan Technical Debt. Atau Technical Debt-nya sudah sangat banyak, jadi *engineer* moralnya turun, jadi Technical Debt-nya prioritasnya sangat banyak. Coding: Terlalu banyak *bug* dan Technical Debt, Moral *engineer* turun karena banyak Technical Debt
K1.24	T: Kemudian kita mau bahas masalah Pair Programming. Biasanya dalam melakukan Pair Programming itu harus mencari orang yang cocok tidak? Supaya tidak ada konflik. N: Disini jarang Pair Programming, kita ga bisa pastiin seberapa seringnya. Konflik pernah terjadi. Karena Pair Programming itu pertama, individunya siap atau tidak. Kedua mereka bisa cocok atau tidak, kayak partner kerja. Kita bisa atau tidak kerja bareng. Kalau tipikal *single fighter*, ga bisa. Karena ga mau nungguin orang. Level juga mempengaruhi. Jika satunya terlalu tinggi levelnya, dan yang tinggi ini bisa mengakomodasi-nya maka akan oke saja. Atau orang barunya yang terlalu minder, susah juga. Coding: Konflik Pair Programming, kesiapan Pair Programming, tipe pekerja yang cocok Pair Programming.
K1.25	T: Dampaknya bagi pekerjaan ini apa yah? Apa dampak Pair Programming? N: jadi lebih *slow*. Coding: Ketidakcocokan Pair Programming memperlambat *development*.
K1.26	T: Frekuensinya? N: Sangat jarang, *so far* kita ga bilang jadi Pair Programming. Jadi saat dibutuhkan saja ngobrol bareng ngelihat *code* bareng-bareng. Kalau bisa jangan dimasukin, takutnya jadi ga relevan. Coding: jarang Pair Programming
K1.27	T: Masalah dokumen *requirementtesting*, Dikurio gimana? N: Nanti kita tanya QA. Kalau secara tim, kita berdasarkan *story*. Jadi *story* ini di-*test*. Coding: test berdasarkan *story*
K1.28	T: Story itu sudah cukup belum untuk menagkomodasi, atau perlu deskripsi lagi?

	N: Tidak perlu. Di Scrum itu, *somehow*, sesuatu itu ga perlu sebegitu *clear*. Contoh, di-*story* kita cuma nulis 3 kata doang. Itu kalau timnya sudah *solid*, mereka sudah tahu apa yang harus mereka lakukan. Tapi dari semua itu, sama 3 kata saja mereka sudah tahu detailnya akan seperti ini, Jadi sudah satu pikiran. Coding: Story tidak perlu terlalu jelas, Tim *solid* sudah *self-organize*.
K1.29	T: Pernah ga *stand upmeeting* jadi ga efektif. Misalnya, tim harus selalu di-oranganize Scrum Master untuk memulai *standupmeeting*. N: Jadi gini, pernah ga efektif. Biasanya kejadiannya gini, ketika ga ada *dedicated* Scrum Master. Jadi contohnya, *case*-nya kita adalah kita punya *captain*. *Captain* itu kan tim internal-nya Scrum-nya langsung. Kadang mereka terjebak diskusi. Tapi tujuannya *daily standup* ga gitu. Plan, *progress* dan hambatannya apa. Diskusinya sebenarnnya *next*-nya. Jadi lama. Kalau semuanya *engineer*, ga masalah, tapi kalau ada PO yang cuma ngelihatin saja. *Captain* jadi Scrum Master, saya kayak ga punya kepentingan untuk mengikuti pembahasan itu, tidak ada *benefit*-nya. Coding: Tidak ada *dedicated* Scrum Master, *Captain*Pengembang pengganti Scrum Master, Daily Standup menjadi ajang diskusi hal teknis, Stakeholder tidak berkepentingan tidak dapat *benefit* pada daily *stand up*.
K1.30	T: Kalau dari frekuensi terjadinya ketidakefektifan Daily Scrum? N: 10%. Jarang banget. Coding: Daily Scrum jarang tidak efektif.
K1.31	T: Dikurio sendiri, ada berapa tim yang implementasi Scrum? N: Kita punya 2 Scrum tim. Tapi, ga selalu Scrum. Contoh, kemarin, tim *mobile* kita ga pakai Scrum karena kita ga punya *backlog*. Jadi isinya *bugs-bugs* saja. Jadi mirip Scrum atau Kanban. Jadi kita cuma punya *priority* mana yang harus di kerjain. Kita ga punya target spesifik. Tapi setiap minggu tetap rilis. Coding: Kehabisan PBI dalam Scrum.

K1.32	T: Berarti, 2 tim bisa beda prosesnya. N: Kalau tim Scrum lagi jalan, bisa beda. Coding: Proses Scrum berbeda antar tim.
K1.33	T: Dampak dia bisa beda ini negatif ga sih? N: Kalau scopenya besar, perusahaan A dan B beda. Contoh, estimasinya ada yang suka *planning* poker, ada juga yang X M L. Tergantungnya yang lebih nyaman gimana. Scrum Master-nya juga bisanya melihat gimana. Contoh paling kecilnya, jadwal *daily standup*-nya beda, terus detail Scrum Board nya bisa beda. Intinya, tujuannya tim nyaman, *velocity* bisa cepat. Coding: Perbedaan Scrum antar perusahaan.
K1.34	T: Definition of done di Kurio gimana? N: Umumnya, si *story* itu. Kita ga punya *definition of done* tertulis. Tim biasanya sudah kebayang jadi kayak apa. Kayak yang tadi saya bilang, dengan 3 kata saja dia sudah tahu jadinya kayak gimana. Karena polanya sudah terbentuk. Contoh, kita mau buat *input* data. Kalau *input* datanya salah, harus ada *feedback*. Kalau itu belum ada, *product*-nya belum jadi. Coding: Tidak ada Definition of Done tertulis.
K1.35	T: Velocity pada awal Scrum, *velocity*-nya rendah. Biasanya bisa di-*track* juga. Ada pengaruh ga *velocity* rendah di awal dan hasil akhirnya? N: Enggak sih. Coding: Tidak ada pengaruh antara *velocity* awal dengan hasil akhir.
K1.36	T: Kemudian, masalah risiko bekerja pada *component team*, apakah tim kurio sudah berorientasi pada *user*. N: Jadi, *story* itu, kita buat *product* untuk *user*. Harus ada dampaknya pada *user*. *Source code* kalian ini adalah sesuatu yang harus di-*maintain*, Jadi *of course* secara *design* harus gampang dilihat, dan *code* mudah dibaca. Coding: Story harus berdampak kepada *user*.
K1.37	T: Semakin banyak anggota tim Scrum malah buat ga efektif. Di kurio gimana? N: Kita pernah ngalamin. Tapi ga pecah jadi tim yang

	beda. Waktu itu kita coba *pairing*. Coding: Pair Programming ketika jumlah anggota tim Scrum banyak.
K1.38	T: Dampaknya dengan semakin banyaknya tim Scrum apa? N: Yang pasti, semakin banyak orang, yang ngomong semakin banyak. Susah buat satu kesepakatan. Komunikasinya juga, jadi lebih banyak *communication channel*-nya. Coding: Jumlah anggota berbanding lurus dengan jumlah *communication channel*-nya.
K1.39	T: Jarang terjadi? N: Tim kita masih kecil, jadi *so far* cukup. Yang *mobile* 2 tim, *non-mobile* 1 tim. Coding: Anggota tim Scrum masih kecil.
K1.40	S: Cara ngatasinya? N: Harusnya sih di-*split*. Saat itu kita belum bisa *split* sama sekali. Jadi *goal*-nya masih satu dan belum bisa di pecah. Coding: Split tim Scrum.
K1.41	T: Ada ga Business Analyst? N: Ga ada. Coding: Tidak ada Business Analyst.
K1.42	T: Perlu ga di Scrum? N: Business Analyst Itu jadinya PO. Tapi, di perusahaan perlu ada orang yang merangkap *role* jadi Business Analyst. Baik PO merangkap jadii Business Analyst. Tapi kurio sendiri belum perlu, tapi kita kedepan mau *plan* untuk punya. Karena kita belum merambah ke bisnis. Kita belum jualan. Coding: Business analyst diperlukan saat perusahaan sudah mau merabah ke bisnis.
K1.43	T: Untuk masalah kurang komunikasi, kira-kira di kurio gimana komunikasinya? Atau pernah terjadi konflik? N: Pernah. Contoh ya, kita punya tim *backend*. Tapi ada *backend* yang software *engineering*. Satu lagi *machine learning*. Nah, mau ga mau, dari sisi kerjaan, kayak air sama minyak, terpisah. Ketika itu terjadi, komunikasinya jadi ga intens.

	Coding: Komposisi tim yang kurang tepat, komunikasi tidak efektif.
K1.44	T: Karena mereka cuma ngerti *skill* mereka masing-masing ya. S: ngatasinnya gimana? N: Kita pernah ada masalah, kan orangnya fokusnya beda, masalahnya adalah saat mau integrasi. Cara ngatasinnya, adalah *earlyintegration*. Jadi kalau sudah diintegrasi di awal, masalahnya ga akan terjadi. Coding: Integrasi di awal.
K1.45	T: Gimana *skill* komunikasi anggotanya, kalau ada anggota yang komunikasi yang kurang? Apakah akan mengganggu ke *development*? N: *By theory*, iya. Tapi di kita ga kejadian. Coding: *Skill* komunikasi penting.
K1.46	T: Dokumentasi *product* ada ga? N: Ga ada. Coding: Tidak ada dokumentasi produk.
K1.47	T: Masalah antar tim, kalau ada lebih dari 1 tim Scrum, gimana koordinasinya? Atau ngerjain kerjaan yang beda. N: *So far* si beda. Tergantung proyeknya. Kita punya UI baru, kita sebut proyek tab. Ada perubahan di *frontend* dan *backend*. Nah, itu kita jadiin 1. Tapi kalau ada kerjaan terkait *machine learning*, tim *machine learning* dan *backend-nya* yang kita satuin. Coding: Pembentukan tim tergantung pada proyek yang akan dijalankan.
K1.48	T: Lalu, masalah *developer* sama QA. Kira-kira perlu kolaborasi gimana? Di kurio gimana? N: Kalau kita *planning*, QA juga ikut *planning* dan kasih bobot. Karena ujung-ujung nya harus *test*. Sedikit banyak mereka akan komunikasi cara *test*-nya. Pengembang akan nyediain beberapa hal untuk *test*. Kita perlu munculin *output*-nya. Gimana cara munculinnya supaya QA bisa *test* atau lihat. Coding: QA terlibat dalam *planning*.
K1.49	T: Terakhir, ketersediaan PO yang hadal. PO nya

	selalu hadir ga di Sprint Planning? N: Kita perencanaannya ada macam-macam. Dulu awal-awal, PO selalu hadir. Biasanya kita punya 2 fase akhir-akhir ini. Jadi ada *planning* untuk manajemen *planning*, ada *product*nya. Ada saya juga yang *proxy*-nya PO kedalam tim. Jadi PO aslinya kayak *client*, si David. Saya jadi PO nya di sisi perusahaan. Kadang, kalau kita ngalamin masalah, dan saya ga punya jawaban untuk *engineer*, saya tarik PO aslinya untuk menjawab. Kan PO aslinya yang tahu prioritasnya. Biasanya sih opsional, mau yang A atau B. Nah itu biasanya, kalau *engineer* kan bisa, mau gampang atau gimana, atau keren atau enggak. Coding: PO mendelegasikan orang lain sebagai *proxy* untuk berhadapan dengan *developer*.

No	**Nama: Hendy Charistianto** **Jabatan: Mobile Lead** **Scrum Role: Pengembang Team**
K2.1	T: Apa risiko menggunakan Scrum dari sudut pandang kak Hendy sebagai Pengembang? N: Kalau risiko kemungkinan lebih kepada PO. Scrum itu metodologi yang mem-*protectdeveloper* dari PO. PO pasti punya beberapa *story* untuk *product*. Kalau ada *feature* yang *critical* untuk *product*, *developer* kan tidak bisa diganggu saat Sprint berjalan. Jadi saat Sprint Planning ga matang, bakalan menghambat. Seharusnya tidak bisa disisipin secara langsung. Feature itu harusnya di*plan* dulu. Coding: Scrum melindungi *developer* dari PO, Sprint Planning tidak matang,
K2.2	S: Bagaimana cara menanganinya? N: Kalau dari kurio sendiri, saat menyisipkan *feature* di tengah Scrum, itu bukan Scrum. Menurut saya, Scrum itu harus mem-*protectdeveloper* supaya fokus. Kan supaya jadi Agile, harus ada *meeting* dan *developer* harus fokus. Coding: Scrum melindungi *developer*
K2.3	T: Jadi masalahnya, Scrum di kurio ga bisa *protectdeveloper*. Pernah ga terjadi penambahan *feature* saat Sprint. N: Kalau dulu belum pernah. Sekarang kita sudah ga terlalu menggunakan Scrum lagi karena tuntutan investor. Kenapa dibilang Agile, sebenarnya gitu.

	Kalau nambah-nambah gitu ga pernah kelar. Coding: Scrum tidak digunakan karena tuntutan investor.
K2.4	S: Kalau di kurio sudah pernah belum nambah-nambah gitu? N: Belum. Kecuali *bugs*, itu kan kritikal. Coding: Penambahan *storybugs* saat Sprint berlangsung.
K2.5	T: Terkait dengan *meeting* Scrum, *developer* sering ga disajak selain *meeting* Scrum? Merasa terganggu ga sih? N: Terganggu banget. Karena ketika *developer* lagi kerja, kayak *pouring all logic inside the brain* dan di-*interupt*, maka buyar. Sama kayak lu ngitung duit. Tiba-tiba di *interupt*. Coding: *Meeting* diluar Scrum mengganggu fokus *developer*.
K2.6	T: *Meeting* yang *interupt* itu diminta pihak manajemen atau bagaimana? N: Iya Manajemen. Manajemen yang bagus itu, *meeting* yang bagus itu harusnya *start or end of the day.* Coding: *Meeting* diminta oleh pihak manajemen, *meeting* seharusnya dipagi atau sore hari.
K2.7	T: Dampak nya malah jadi ga konsen. N: Ya. Coding: Pengembang menjadi kehilangan konsentrasi akibat dari *meeting*.
K2.8	T: Sering ga terjadi kayak gitu? N: Dulu enggak, sekarang sering. Karena kita sudah ga *pure* Scrum lagi. Coding: Penerapan Scrum yang tidak sesuai lagi.
K2.9	T: Cara ngatasi supaya ga buyar? N: Setiap *end of* Scrum kan ada Retrospective, nah kita nulis *angry*-nya kenapa, biasanya *too muchmeeting*. Scrum Master itu kan harusnya jagain *developer* juga. Coding: Membahas masalah *too muchmeeting* di Retrospective.

K2.10	T: Berkaitan dengan Sprint Retro, Sprint retro kan selalu di akhir. Sampai sekarang, di kurio pasti diadakan ga di akhir Sprint? N: Ga rutin. Biasanya ketika Scrum Master ga ada, intensitas Scrum Method mulai berkurang. Tugas Scrum Master itu mengatur supaya kegiatan Scrum integritasnya tetep ada. Coding: Sprint Retrospective tidak dilaksanakan dengan rutin, Scrum tidak dijalankan dengan baik tanpa Scrum master khusus.
K2.11	T: Ada ngerasain dampak ga sih dari perubahan ini? Ga ada Scrum Master dan jarang Retro? N: Dampaknya pasti kerasa, pas *development* itu *stress* lebih tinggi. Pengembang jadi ga fokus. Butuh waktu lama untuk berada di dalam *state* dimana dia benar benar fokus. Jadi kerjaannya ga teratur dan jadi lamban. Coding: Pengembang tidak fokus, ketidakadaan Scrum master.
K2.12	S: Misalkan lagi mulai Sprint, ada ga masalah pribadi antar *developer*. N: Mungkin ada, karena setiap Scrum, setiap individu diharapkan memiliki *self-conciousness*. Jadi biasanya, Scrum berjalan dengan baik tergantung dari individu tim. Jadi kalau *solid* dan bisa tutupin *task* masing-masing, bisa lebih cepat. Beda sama individu yang karena Scrum malah males-malesan. Jadi lamban dan gabut. Malah bisa jadi *slack* gitu. Coding: Ada masalah pribadi dalam Scrum, individu harus memiliki kesadaran diri.
K2.13	S: Ada ga sih kasus gitu di Kurio? Cara ngatasinya kalau terjadi gitu? N: Ada sih. Biasanya sih *one on one*, *speak*. Itu kalau misalnya dia berani ngomong. Biasanya lewat *Retrospective*. Atau ngomong ke atasan atau Scrum Master. Coding: Menyelesaikan masalah pribadi dengan berkomunikasi empat mata, mengungkapkan permasalahaan di *Retrospective*, membicarakan masalah dengan Scrum Master.
K2.14	T: Tujuan proyek kurang jelas. Di kurio gimana?

	N: Disini ada visi misi. Biasanya setiap *planning* yang baik, tujuannya mau ngapain. Misalnya kita kan *data driven*. Kita mau ningkatin *retention rate by* UI. Tapi ga selalu visi misi nya dijelaskan. Coding: Tujuan selalu dijelaskan pada *planning*.
K2.15	T: Dengan adanya visi misinya tujuan proyek jadi jelas? N: Biasanya kita ga terlalu sih. Kita biasanya sih jadi eksekutor doang. Tapi ga ngerti intinya itu apa. Coding: Visi misi belum cukup untuk memperjelas tujuan proyek, *developer* tidak mengerti tujuan proyek.
K2.16	T: Dari ketidaktahuan intinya, apa jadi ada dampak apa ga bagi *developer*? N: Biasanya kita *develop* ga sesuai ekspektasi PO. Coding: Kurang jelasnya tujuan proyek, hasil yang tidak sesuai ekspektasi PO.
K2.17	T: Nah, itu sering terjadi? N: Jarang. Coding: Jarang terjadi tujuan proyek tidak jelas.
K2.18	T: Dari kurio sendiri, ada cara ngatasin ga sih supaya inti nya tersampaikan? N: Biasanya, pas ScrumPlaning, ketika desain sudah jadi, kita rembukan bareng. Saling kasih ide masing-masing. Kita mengungkapkan *why*. Lalu dari *developer* sendiri, kita kasih tahu *limitation* kita. Coding: Mendiskusikan tujuan proyek pada saat Sprint Planning.
K2.19	T: Requirement yang tidak jelas. Pernah terjadi ga sih? N: Selama ini sih enggak ya. Coding: Kebutuhan cukup jelas.
K2.20	T: Disini ada risiko konflik antar 2 PO. N: Disini ga ada karena cuma 1 doang. Coding: -
K2.21	T: Prioritas *requirement*-nya gimana sih disini?

	N: Biasanya sih kita rembukan bareng PO. Biasanya PO tentuin mana yang mau *release* duluan. Itu yang jadi prioritas tertinggi. Di Scrum kan ada yang namanya *weight* untuk setiap *story*. Biasanya PO sendiri yang nimbang, dari pada *feature* ini ga keburu, mending yang lain dulu di-*develop*. Coding: PO menentukan prioritas PBI.
K2.22	T: Kalau perubahan *requirement*, pernah terjadi ga? N: Kalau kayak gitu sih belum pernah. Coding: Belum pernah terjadi perubahan kebutuhan.
K2.23	T: Atau detail UI-nya berubah. N: Biasanya kita lanjutin di *next* Sprint. Coding: Perubahan UI produk dimasukan ke *story* pada Sprint selanjutnya.
K2.24	T: Disini pernah ngadain Technical Debt ga sih? N: Ada, biasanya Technical Debt itu di *refactoring*. Sekarang kita juga lagi *refactoring* 3 minggu. Biasanya bukan berdasarkan bobot tapi *timebased*. Coding: Melakukan *refactoring* di Technical Debt.
K2.25	T: Berarti Technical Debt tidak dimasukan ke Sprint? N: Enggak, biasanya di end of Scrum. Disela-sela *project* baru. Baru ada *refactoring*. Coding: *Technical Debt* tidak dimasukan kedalam Scrum.
K2.26	T: Bagaimana proses Technical Debt-nya disini? Ada *meeting* ga? N: Biasanya kita ada *meeting* kecil. Dari *lead*-nya sendiri. Nentuin apa saja yang harus di-*refactor*. Kita ngelihat *code* biasanya. Berdasarkan *agreement* juga. Kalau dalam tim kan ada *codeagreement*. Nentuin *architecture*-nya pakai yang mana. Coding: Melakukan *meeting* untuk *refactoring*.
K2.27	T: Di kurio sendiri ada Pair Programming ga? Pernah ga ada ketidakcocokan dalam Pair Programming? N: Pair Programming ada. Ketidakcocokan pastinya ada. Tapi balik lagi ke *codeagreement*. Jadi *agreement* nya harus dipatuhi.

	Coding: Ada ketidakcocokan saat Pair Programming, Pair Programming berdasarkan *codeagreement*.
K2.28	T: Bagaimana jika ketidakcocokan ini berdasarkan *pesonal*nya? N: Personal sih ada biasanya. Coding: Ada permasalahan *pesonal* saat Pair Programming.
K2.29	T: Kalau terjadi kayak gitu dampaknya apa? N: Dampaknya, ga bisa kerja sama dengan baik. Kalau kita ngatasi itu dari *interview*. Kita ngenalin semua orang ke tim. Jadi kita kasih pertanyaan. Jadi kalau ga cocok kita ga terima. Ini *interviewengineer*. Dan kalau misalnya kriteria nya ga cocok satu sama lain, kita keluarin. Coding: Mengatasi ketidakcocokan dari *interview* kerja.
K2.30	T: Sering ga terjadi ketidakcocokan ini? N: Jarang sih. Biasanya kita sudah saring lewat *interview* dulu. Biasanya sih ga lolos. Coding: Jarang terjadi ketidakcocokan antar *developer*.
K2.31	T: Pengembang ada melakukan *unit testing* ga? Dan butuh dokumen *requirement* ga? N: Ada *unit testing*. Ga butuh dokumen. Kita berdasarkan *cleanarchitecture* saja biar gampang *test*-nya. Kita ga ada dokumentasi apapun kecuali API. Coding: *Unittesting* tidak menggunakan dokumen, *unit testing* berdasarkan *cleanarchitecture*, ada dokumentasi API.
K2.32	T: Masalah *standupmeeting*, sudah efektif belum? Mungkin *standupmeeting* yang harusnya sebentar malah jadi lama. N: Kalau kita sih efektif. Standup *meeting* ga lebih dari 10 menit. Kalau ada *tech difficulties*, ngomong sebentar, lalu rembukan setelah *meeting*. Kalau *out of topic* sih enggak. Coding: Standup *meeting* efektif, melanjutkan bahasan teknis diluar *standupmeeting*.
K2.33	T: Ada berapa tim yang implementasi Scrum di Kurio?

	N: Kita biasanya *mobile* ada sendiri, *backend* ada sendiri, *machine learning* ada sendiri. Coding: Penyusunan tim Scrum berdasarkan bidang keahlian.
K2.34	T: Kalau dari setiap tim tersebut, ada perbedaan ga implementasi Scrum nya? N: Biasanya sih sama. Karena Scrum Master kita biasanya cuma 1 doang. Jadi tergantung Scrum Master. Yang jadi masalah adalah *dependency*. Coding: Implementasi Scrum dalam perusahaan tergantung pada Scrum Master.
K2.35	S: Kalau terjadi gitu, gimana ngatasinya? N: Biasanya, yang *dependent* gitu, kita *drop* dan taruh di *next*Sprint. Coding: Story yang *dependency* dimasukan ke *next*Sprint.
K2.36	T: Penyebab nya gimana? N: Karena Scrum Board-nya pisah-pisah. Contohnya *mobile* butuh API dari *backend*. *Mobile* ga bisa kerjain kalau belum ada API-nya. Coding: Scrum *board* yang terpisah
K2.37	T: Dampaknya? N: Kita biasanya kerjain *nextstory*-nya. Coding: *Dependency* menyebabkan *developer* mengerjakan *story* yang tidak *dependent* terlebih dahulu.
K2.38	T: Sering terjadi ga? N: Lumayan sering. Tapi ga selalu. Kalau Scrum Masternya bagus, *backend* duluan Scrum nya baru *mobile*. Jadi ga *dependent*. Coding: Sering terjadi *dependency*.
K2.39	T: Kurio punya *definition of done* ga? N: Biasanya setelah *codereview* dan di *test* QA. Coding: Adanya *definition of done*.
K2.40	T: Berarti semua Scrum harus ikuti aturan ini. Kalau *velocity* yang lebih rendah di awal gimana? Ngaruh ga ke keseluruhan proses Scrum?

	N: Kita kan tentuin *task* berdasarkan *velocity*. Kita kan ada *burn down chart* juga, kita bisa lihat disana. Kalau *velocity* kita lebih tinggi dari *task*-nya, jadi kebanyakan bengong. Awal-awal biasanya agak ngaco. Coding: *Burn down chart*, *Velocity* awal tidak tepat.
K2.41	T: Kok bisa ngaco? N: Lack of *experience* dari *developer* atau tim baru yang belum bisa ngukur *velocity*. Coding: Kurangnya pengalaman *developer* dalam mengukur *velocity*.
K2.42	T: Dampak? N: Kalau *velocity*-nya rendah, *task* lebih sedikit, ya jadi banyak bengong. Beda halnya kalau ngukur *velocity*-nya ketinggian, malah jadi kelabakan. Coding: *Velocity* harus sesuai dengan kemampuan *developer*, *velocity* rendah maka *task* sedikit, *velocity* tinggi maka *task* banyak.
K2.43	T: Sering terjadi di kurio? N: Awal-awal sih kita *velocity* agak ngaco di2 Sprint pertama. Tapi setelah nya sudah nyesuain. Coding: Velocity awal tidak tepat.
K2.44	T: Kalau di kurio sendiri, lebih penting acuan ke *customervalue*, atau lebih mentingin *code* yang lebih *clean*. Atau ada hubungannya? N: *Balance* sih ya. Kita pasti mau nge-*pushfeature* yang *uservalue*-nya ada. Kita berusaha *estimate* dari bobot masing-masing *planning*. Kita hitung juga dari *code* yang kita hasilkan. Jadi kompleksitasnya juga diperhitungkan saat *planning*. Supaya ga terlalu berantakan dan *customervalue*-nya ada. Coding: *Customervalue* dan *cleancode* sama pentingnya.
K2.45	T: Semakin banyak anggota Scrum. Mengganggu ga ya? N: Mengganggu. Karena semakin banyak *prorgammer*, semakin banyak *logic*, semakin banyak variasi di dalam *code*. Kalau masing-masing *prorgammer* belum bisa bekerja dengan baik, maka akan mengganggu. Coding: Terlalu banyak anggota tim Scrum mengganggu

	kinerja *prorgammer*.
K2.46	T: Ganggu nya gimana? N: Dibagian komunikasi ada. Terus aplikasi-nya *bug*gy. Lebih banyak *bug*. Karena variasi *ofcode*. *Redundantcode*. Coding: Terlalu banyak anggota tim Scrum mengganggu komunikasi dan membuat produk aplikasi lebih banyak *bug*.
K2.47	T: Kalau di kurio, ada ketentuan khusus ga timnya harus berapa? Atau kebanyakan. N: Belum ada ketentuan. Tapi biasanya, kita batasin 3 di 1 divisi. Tapi kedepannya, kita pecah lagi per *project* yang mau di-*build*. Coding: -
K2.48	T: Berarti jarang terjadi masalah kebanyakan? N: Kita kekurangan orang. Coding: Masih kekurangan sumber daya manusia
K2.49	T: Business Analyst ada ga? N: Merangkap dari PO dan Data Analyst. Coding: Tidak adanya Business Analyst.
K2.50	T: Ada dampaknya ga? Atau perlu ga Business Analyst selain Product Owner? N: Menurut gue sendiri perlu. Karena *specialty*-nya beda. Coding: Perlu adanya Business Analyst.
K2.51	T: Atau Business Analyst-nya perlu ditentukan apakah kurio mau ke bisnis atau enggak? N: Mungkin ada pengaruh nya saat mau *getprofit*. Coding: Business analyst diperlukan jika perusahaan mau mendapatkan profit.
K2.52	T: Kurangnya komunikasi antar anggota tim. Ada ga di kurio? N: Kalau dari *developer* sih, komunikasi dilakukan saat ada *codereview*. Jadi saat orang mau *build* suatu *feature*, pasti ada pull *requirement* di Github, lalu kita *review*. Eh ternyata *method* ini sudah ada yang bikin.

	Coding: Komunikasi *developer* dilakukan pada *codereview*.
K2.53	T: Sudah cukup belum *codereview* saja untuk komunikasi. N: Cukup dari segi *developer*. Kita juga ada Scrum*meeting*. Coding: Code *review* cukup untuk melakukan komunikasi *developer*.
K2.54	T: Kalau masalah *skill* komunikasi ngaruh ga? N: Ngaruh, dan ada di kurio. Cara atasinya, susah juga karena bawaan individu. Kalau *developer*nya kita komunikasikan dan bisa *improve*, ya bagus. Kalau ga bisa, kita *ignore* sampai dia keluar. Coding: *Skill* komunikasi mempengaruhi proses *development*.
K2.55	T: Dampaknya kalau ada orang kayak gitu? N: Ga terlalu dampak besar. Kalau ga bagus komunikasinya, ya di-*cover*. Coding: Komunikasi yang kurang tidak memberikan dampak yang berarti.
K2.56	T: Jarang ya ada yang kayak gitu? N: Lumayan sering sih. Biasanya komunikasiin sebentar. Kalau tetap melakukan kesalahan yang sama saat komunikasi ya kita mau ngapain? Coding: Sering ada *developer* yang memiliki kemampuan komunikasi yang kurang.
K2.57	T: Kalau masalah *product*, setiap Sprint kan menghasilkan suatu *product*. Perlu ga dokumentasi untuk masing-masing *feature*. N: Kalau menjunjung tinggi *maintenance*, perlu. Tapi karena kurio masih belum bayak *developer*, kita belum. Coding: Dokumentasi untuk membantu *maintenance*, dokumentasi belum diperlukan apabila jumlah *developer* sedikit.
K2.58	T: Kalau masalah komunikasi antar tim, itu bagaimana koordinasiinnya? N: Biasanya lewat Project Manager dan Scrum master, kita komunikasi lewat mereka. Biasanya orang yang berkaitan ditarik untuk *discuss* bareng.

	Coding: Koordinasi masalah komunikasi melalui Scrum master.
K2.59	T: Kayak gitu sudah cukup? N: Cukup karena *engineeringmanager*-nya juga punya *techknowledge*. Coding: Koordinasi dibantu oleh *engineeringmanager* yang mempunyai *technicalknowledge*.
K2.60	T: Kalau kolaborasi antar *developer* dan QA gimana? N: Kalau saya sendiri sih masih kurang. Coding: Kurangnya kolaborasi antar *developer* dan QA.
K2.61	T: Apa penyebabnya? N: Mungkin karena QA belum digunakan secara baik. QA ga selalu Scrum kita. QA kan seharusnya ngetest app kita ada *bug* atau enggak. Tapi kadang masih ada *bug*. Coding: Kurangnya keterlibatan QA dalam Scrum.
K2.62	T: Dampaknya? N: App-nya ada *bug*. Coding: Kurangnya keterlibatan QA dalam Scrum memicu aplikasi yang *bug*gy.
K2.63	S: Cara mengatasinya? N: Seharusnya *unit testing* saja cukup sih. Tanpa ada QA. Karena lebih cepat. Coding: Unit *testing* tanpa QA.
K2.64	T: Sering ga terjadi kolaborasinya kurang antar QA dan Dev nya? N: Terjadi. Coding: Kolaborasi QA dan *developer* kurang baik.
K2.65	T: PO nya sudah handal belum? N: Handal sih. Dia tahu visi misi nya dan tahu mau apa saja yang dibuat. Coding: PO sudah handal.

No	**Nama: Steven Tiolie** **Jabatan: Software Engineer**

	Scrum Role: Development Team
K3.1	T: Apa risiko menggunakan Scrum? N: Kalau dari Scrumnya sendiri. Satu, *developer* ga bisa milih *story*. Karena semua telah diurutin berdasarkan *priority*-nya. Coding: Pengembang tidak bisa memilih *story*.
K3.2	T: Dampaknya *developer* ga bisa milih *story*? N: Kadang ada *developer* yang suka ngerjain ini tapi Scrum memaksa kerjain prioritas dulu. Coding: -
K3.3	S: Terus kalau kayak gitu menanganinya gimana? Kan sudah diurutin dan ga bisa pilih. N: Mau ga mau. Misal, Sprint 1 kita familiar sistem *login*. Sprint 2 kita kerjain yang lain tapi orang lain kerjain *login*. Jadi solusinya harus siap diganggu untuk komunikasi gimana kerjainnya. Coding: Pengembang harus siap diganggu untuk komunikasi saat Sprint.
K3.4	T: Dikurio sendiri berapa kali *meeting*? N: KPI *Meeting* setiap hari, *meeting* khusus tim 1 minggu sekali, Itu yang diluar *stand upmeeting* dan Sprint Planning. Coding: Banyak *meeting* diluar Scrum.
K3.5	T: Kalau *meeting* diluar Scrum mengganggu ga? N: Enggak, bentar doang 15 menit saja. Nunjukin *achievement* kita hari itu. Coding: *Meeting* diluar Scrum tidak mengganggu karena durasinya singkat.
K3.6	T: Sprint Retrospective-nya gimana? Sering ga? N: Sering dulu pas masih di tim *mobile*. Coding: -
K3.7	S: Sekarang gimana? N: Sekarang, sudah 2 atau 3 bulan ga ada. Coding: Retrospective tidak jalan.
K3.8	T: Biasanya efektif ga? N: Efektif sih. Tapi yang didenger kata-kata yang

	berulang-ulang. Setiap kali retro isinya Sprint kelar, app sudah *publish*. *Too muchmeeting*. Coding: Retrospective itu efektif, pembahasan *Retrospective* monoton.
K3.9	T: Sekarang kan jarang retro, ada perngaruhnya atau dampaknya? N: Ga terlalu berasa sih. Sebenarnya itu cuma buat introspeksi diri saja sih. Coding: Tidak adanya *Retrospective* tidak terlalu berdampak pada anggota tim Scrum.
K3.10	S: Sering terjadi masalah pribadi di kurio? N: Jarang sih kalau yang saya rasain. Kalau saya sih lancar-lancar saja. Coding: Jarang terjadi masalah pribadi dalam Scrum.
K3.11	S: Kalau ada masalah pribadi antar *developer*, cara mengatasinya bagaimana? N: Kurang ngerti sih kalau masalah itu. Coding: -
K3.12	T: Tujuan proyek kurang jelas. Gimana? N: Menurut saya, dengan menggunakan Scrum, tujuannya jadi jelas. Diawal ada didefinisikan. Coding: Tujuan proyek jelas.
K3.13	T: Kalau *requirement*nya jelas ga? N: Biasanya jelas di awal. Tapi biasanya lupa jadi sering tanya-tanya. Coding: Kebutuhan jelas, *developer* sering lupa kebutuhan.
K3.14	S: Kalau lupa gimana? N: Tanya saja ke PO nya lagi. Coding: Pengembang bertanya ke PO apabila lupa kebutuhan-nya.
K3.15	T: Kalau prioritas *requirement*-nya. Kalau dari sisi *developer*nya pernah ngerasain ga salah prioritas? N: Itu biasanya didebatin di awal. Cuma kadang-kadang kalau lagi capek, ya dibiarkan diawal. Tapi ditengah malah debat.

	Coding: Prioritas kebutuhan tidak memadai.
K3.16	T: Dampaknya apa? N: Kalau estimasinya ga akurat, tadi nya ada 6 *story*, tapi yang berhasil dikejar cuma 5, tapi *story* yang penting malah ga kekejar. Coding: Story penting tidak selesai.
K3.17	T: Frekuensi terjadi *missed*-nya gimana? N: Medium Coding: Jarang terjadi salah estimasi.
K3.18	T: Kalau masalah perubahan *requirement*, pernah terjadi ga? N: Pernah, tapi jarang banget. Coding: Jarang terjadi perubahan kebutuhan.
K3.19	T: Kenapa bisa terjadi kayak gitu? N: Lupa. Coding: -
K3.20	S: Kalau berubah, itu gimana *developer*nya? N: Tetap lanjut sih. Kadang bisa di-*estimate* ulang. Atau di-drop *story*-nya. Coding: Estimasi ulang untuk perubahan kebutuhan, melakukan drop *story*.
K3.21	T: Dampaknya? N: Ada *story* yang ga kelar. Coding: Perubahan kebutuhan menyebabkan *story* tidak selesai.
K3.22	T: Pernah Technical Debt? N: Pernah. Coding: Pernah melakukan Technical Debt dalam Scrum.
K3.23	T: Sudah pas belum Technical Debt-nya, atau sudah efektif belum? Di-oranganize sendiri oleh *developer*-nya? N: Biasanya kita *developer* inisiatif sendiri untuk masukin kedalam *story* atau dibuat parallel. Jadi nambahin *story* yang isinya Technical Debt.

	Coding: Pengembang memasukan Technical Debt kedalam *story*.
K3.24	T: Pair Programming pernah? Ada masalah ketidakcocokan *code* atau ada masalah individu yang tidak cocok? N: Kalau *pairing* sih, kita sudah mengikuti *standard* di awal. Jadi harus ikuti standard. Coding: Pair Programming memiliki standard yang harus diikuti.
K3.25	T: Pernah ga merasa cocoknya sama orang tertentu. N: Pernah. Coding: Terjadi ketidakcocokan dalam Pair Programming.
K3.26	T: Kalau dipasangin sama pasangan yang kurang cocok gimana? N: Dampak ke kerjaan sih engggak, paling kalau kurang akrab sama partner, ya ga bisa sambil bercanda. Coding: Tidak ada dampak ketidakcocokan Pair Programming pada pekerjaan.
K3.27	S: Ga bisa pilih ya? N: Pairing sih kita ga ada aturan yang *strict*. Jadi bisa sama siapa saja. Kalau lagi butuh bantuan. Coding: Dapat memilih pasangan Pair Programming.
K3.28	T: Tapi kalau di-*pairing* sama orang yang ga cocok jarang ya? N: 50 50 sih. Coding: Kadang-kadang *developer* dipasangkan dengan orang yang tidak cocok dalam Pair Programming.
K3.29	T: Pernah *unit testing*? N: Belum Coding: -
K3.30	T: Perlu dokumen ga untuk *unit testing*? N: Paling dari kita sendiri saja. Ga ada dokumen khusus. Coding: -
K3.31	T: Terus masalah *stand upmeeting*nya sudah efektif

	belum? N: Selama ini ya efektif-efektif saja. Tapi sudah mulai kayak rutinitasnya. Coding: Standup *meeting* efektif.
K3.32	T: Berarti *standupmeeting* selama ini ga ada keluhan lagi? N: Terlalu lama. Kadang-kadang bisa sampai 30 menit. Coding: Standup *meeting* terlalu lama.
K3.33	T: Penyebabnya apa? N: Kadang kadang ada ngebahas *technicaldetail* di-*daily standup*. Dan dampaknya jadi molor. Coding: Membahas hal teknis adalah penyebab Daily Scrum terlalu lama.
K3.34	T: Cara mengatasinya? N: Biasanya pikiran sudah dimana-mana, dan berharap cepat selesai. Coding: Pengembang tidak fokus pada Daily Scrum
K3.35	T: Proses Scrum di setiap tim berbeda ga? N: Sama. Coding: Proses Scrum di tiap tim sama.
K3.36	T: Definition of donenya gimana? N: Kan kita ada on progress, *review* dan *done*. Jadi di-*review* itu make sure kalau ga ada potential *bugs*. Coding: Definition of done jelas.
K3.37	T: Kalau *velocity* tim itu, kalau diawal rendah? N: Diawal malah tinggi. Jadi ya malah salah *estimate*. Coding: Salah *estimatevelocity* diawal.
K3.38	T: Ditanamkan ga sih fokus ke *customervalue*? N: Kita sih ngerjain *story* yang di-*define* sama PO. Jadi cuma eksekutor saja. Coding: Pengembang hanya merasa sebagai eksekutor.
K3.39	T: Kalau masalah jumlah tim Scrum yang makin

	banyak. N: Tim kita masih sedikit. Coding: -
K3.40	T: Business analyst, perlu ga? N: Kurang tahu juga sih. Yang jelas ada sih perlu, karena bentar lagi mau *monetize*. Coding: Business analyst diperlukan saat sudah mau monetize.
K3.41	T: Masalah komunikasi antar anggota tim, ada masalah ga? N: Lancar-lancar saja. Coding: Tidak ada masalah komunikasi antar anggota tim.
K3.42	T: Kalau *skill* komunikasi individu yang kurang? N: Ada, tapi belum pernah ngadepin. Dari karakteristiknya. Tapi ga ngaruh, karena dikasih tugas tetap dikerjain. Coding: *Skill* komunikasi yang kurang dari *developer* tidak mempengaruhi *development*
K3.43	T: *Product*-nya perlu dokumentasi ga? N: Harusnya perlu, tapi kita belum. Coding: Perlu adanya dokumentasi *product*.
K3.44	T: Kenapa belum ada dokumentasi? N: Pada malas, apalagi bagi *developer*. Coding: Pengembang malas membuat dokumentasi.
K3.45	T: Ada ga dampak dari belum ada dokumentasi produk. N: Kalau ada anak baru, bisa suruh baca dulu baru jelasin. Coding: Tidak adanya dokumentasi membuat proses inisiasi anggota baru menjadi lebih lama.
K3.46	S: Kalau gitu gimana? N: Ujung-ujungnya ada proses *onboarding* dari CEO-nya. *Meeting* jelasin produk kita ngapain. Tapi kurang efektif. Kalau pakai baca kan kapan saja dia lupa bisa buka lagi.

	Coding: Diperlukan proses *onboarding* untuk karyawan baru mengenai produk.
K3.47	T: Pernah ga terjadi *overlap* atau *dependency*? N: Lumayan sering. Coding: Sering terjadi *dependency*.
K3.48	T: Kenapa bisa terjadi *dependency*? N: Ekspektasi data API di *mobile* dan *backend* berbeda. Paling kejadian begitu sering dan jadinya ngobrol lagi untuk menyesuaikan. Coding: *Dependency* disebabkan karena kurangnya komunikasi antar tim.
K3.49	T: Untuk koodinasi dengan QA? N: Belum pernah koordinasi sama QA saya. Paling cuma kayak *user* biasa gitu koordinasinya? Coding: -
K3.50	T: PO-nya sudah handal? N: Kita sudah beberapa kali ganti Product Owner. Sekarang handal karena representasi kantor dari jepang. Coding: PO sudah handal.
K3.51	T: Sebelum-sebelumnya pernah kurang handal? N: Dulu, PO-nya itu belum bisa ukur semuanya secara kuantitatif. Coding: PO belum bisa mengukur secara kuantitatif.
K3.52	T: Dampaknya ke *development*? N: Paling dampaknya kita ga bisa mengukur dengan tepat efek dari penambahan *feature* tersebut, *retentionrate*-nya dan berapa *user* yang nambah. Coding: Kurang handalnya PO menyebabkan *developer* tidak bisa mengukur dengan tepat efek dari penambahan *feature*.

No	**Nama: Sella** **Jabatan: UI Designer** **Scrum Role: Development Team**
K4.1	T: Apa risiko penggunaan Scrum? N: Justru pakai Scrum malah lebih jelas. Kayanya

	belum ada deh. Coding: -
K4.2	T: Sering ga sih ada *meeting* selain *meeting*Scrum? N: Kadang-kadang sih buat ngebahas *design*. Coding: Ada *meeting* tambahan diluar Scrum.
K4.3	T: Ganggu kerjaan ga? N: Ga. Malah membantu. Jadi bahas *design*. Coding: *Meeting* diluar Scrum tidak mengganggu pekerjaan.
K4.4	T: Sprint retro. Ada ga? Dan berjalan dengan baik ga? N: Lumayan baik. Coding: Sprint retro berjalan baik.
K4.5	T: Kalau tidak dijalankan ada dampaknya? N: Kayaknya ada, kita kan jadi evaluasi. Mungkin hal ini kurang berjalan dengan baik. Dan bagus-bagus nya juga apresiasi gitu ke tim. Coding: -
K4.6	T: Ada ga yang ngekritik kerjaan orang di retro? Ada yang sampai bikin sakit hati ga? N: Ga tahu sih. Kan bukan aku yang kena dan ga bisa ngomong untuk orang lain. Coding: -
K4.7	T: Pernah ada masalah pribadi di Scrum? N: Setahu aku sejauh ini ga ada sih. Coding: Tidak ada masalah pribadi.
K4.8	T: Biasanya, Sprint Planning jelasin tujuan proyek ga? Atau tujuannya ga jelas? N: Jelas sih, karena untuk buat *design* perlu jelasin tujuannya.Biasanya PO-nya cerita mau gimana nih. Coding: Tujuan proyek jelas.
K4.9	T: Pernah ga sih terjadi *requirement* untuk *design*-nya ga jelas? N: Iya kadang-kadang.

	Coding: Kadang terjadi kebutuhan desain yang tidak jelas.
K4.10	T: Penyebabnya apa nih? N: Desainkan subjektif gitu. Kadang-kadang suka disitunya sih. Coding: Kebutuhan desain kurang jelas karena desain bersifat subjektif.
K4.11	S: Cara tanganinya? N: Kita *eksplore* dan buat *alternative*. Harusnya kan desain lalu *test* langsung ke *user*nya. Jadi dikira-kira mungkin *user* suka yang kaya gini, tapi juga bisa yang kayak gitu. Coding: Melakukan eskplore dan membuat beberapa desain sebagai *alternative*.
K4.12	T: Dan dampaknya malah jadi harus buat *alternative*. N: Iya, jadi produknya harus direvisi terus *design*-nya. Coding: Revisi desain.
K4.13	T: Pada Sprint Planning, prioritasnya sudah memadai belum sih? N: Belum bisa jawab karena selama ikut belum ada prioritas. Coding: -
K4.14	T: Pernah ga sih terjadi perubahan *requirement*? N: Pernah, waktu itu pernah terjadi karena lihat *statistic*. Kemarin kan pakai asumsi. Tapi lihat data malah beda. Padahal *design*-nya sudah ½ jalan. Coding: Pernah terjadi perubahan kebutuhan, perubahan kebutuhan berdasarkan statistic.
K4.15	S: Cara atasinya? N: Ikuti saja. Coding: -
K4.16	T: Dampaknya harus *design* ulang? Dan sering? N: Cuma rombak saja sih. Jarang terjadi. Coding: Jarang merubah desain.
K4.17	T: Stand up *meeting* sering ikut ga?

	N: Dulu sering, sekarang ga pernah. Coding: Desainer jarang mengikuti Daily Scrum.
K4.18	T: Menurut kakak sudah efektif? N: Sebenarnya efektif, tapi kalau dari *design* kan bikin *eksplore*-nya banyak dan berhari-hari. Jadi ya hari ini masih sama, besok juga masih sama. Tapi ya perlu datang juga sih. Coding: Daily Scrum belum terlalu efektif untuk desainer.
K4.19	T: Dampaknya bagi desainer jika belum ada progress apa? N: Ga masalah sih, paling 15 menit. Coding: Ketidakadaan *progress* dari desainer tidak menjadi masalah pada Daily Scrum.
K4.20	T: Tapi sering terjadi gitu? N: Lumayan. Kalau pas lagi kerjain beda-beda, setiap hari bisa berbeda. Coding: Sering terjadi Daily Scrum yang kurang efektif.
K4.21	T: Terus cara mengatasi hal itu? N: Ya bilang saja masih kerjain yang kemarin. Ikut saja dengerin. Coding: Menyampaikan *progress* desain di Daily Scrum.
K4.22	T: Kalau diKurio sendiri, *done* itu definisi nya apa? N: *Done* itu kalau sudah selesai dan di-*approve* sama PO. Coding: Ada *definition of done* yang jelas.
K4.23	T: Desain pakai *velocity*? N: Enggak pernah pakai. Coding: -
K4.24	T: Kalau masalah jumlah anggota tim Scrum, ngerasa semakin efektif ga? N: Kalau kebanyakan malah ga perlu, perwakilan saja.

	Coding: -
K4.25	T: Kalau Business Analyst perlu ga? N: Perlu sih, karena *projectrequest* kan bisa dari *analyst*. Kalau ada masalah bisa solusinya dari desain. Coding: Perlu adanya Business Analyst
K4.26	T: Kalau sekarang tanpa BA bisa jalan? N: Bisa jalan. Coding: -
K4.27	T: Kalau masalah komunikasi, di Kurio komunikasinya gimana? N: Oke sih. Coding: Tidak ada masalah komunikasi.
K4.28	T: Kalau masalah *skill* komunikasi, ngaruh ga kalau ada yang kemampuan komunikasinya kurang? N: Pintar-pintarnya kita menyesuaikan diri pada karakter setiap orang. Coding: Penyesuaian diri terhadap karakter anggota tim.
K4.29	T: Dampaknya kalau ada orang yang susah komunikasi? N: Ga terlalu berdampak. Kitanya jadi harus lebih aktif. Kalau sudah lempar desain, tapi 2 atau 3 hari ga ditanya, jadi kitanya yang harus aktif tanya. Coding: Kurangnya kemampuan komunikasi seorang anggota menuntut anggota lain untuk lebih aktif.
K4.30	T: Kalau PO-nya sudah handal belum? N: Dulu sih jelas ada Product Manager, tapi dia *resign*. Jadi sekarang masih kosong. Pak David yang jadi PO. Sekarang sih ada VP Product, jadi *technically* dia PO nya. Coding: Tidak ada *dedicated* PO

No	**Nama: Arie** **Jabatan: Senior API Engineer** **Scrum Role: Development Team**
K5.1	T: Apa risiko menggunakan Scrum?

	N: Kita ga tahu the whole *requirement*. Selama beberapa Sprint kita belum tahu the whole *product* nya seperti apa. Terus, dari segi *engineering*, bisa saja dia *deliver* ga sesuai dengan beberapa *story* yang ditetapkan. Mungkin kita berpikir komitmen kita ga sesuai dengan harapan. Coding: Pengembang tidak mengetahui kebutuhan secara keseluruhan, Pengembang mengerjakan *story* yang tidak sesuai dengan yang ditetapkan.
K5.2	T: Kesalahan estimasi? N: Iya, atau kita ga tahu *velocity*. Coding: Kesalahan estimasi terjadi karena belum mengetahui *velocity* tim.
K5.3	T: Ga tahu *velocity* itu malah jadi penyebabnya? N: Biasa jadi. Pertama kali pakai Scrum, gue *totallyblank* berapa *velocity* gue. Risiko pertama kali pakai Scrum, dia ga tahu *velocity*, dan bisa *failed* atau ga ke-*deliver* semua *story*-nya. Coding: Tidak bisa mengukur *velocity* saat pertama menggunakan Scrum.
K5.4	T: Kalau sudah seperti gitu, ada cara mengatasinya ga supaya bisa tahu *velocity*? N: Kalau menurut gue sih pasti ada kemungkinan *failed* untuk pertama kali, karena kita ga tahu *velocity* kita. Kecuali *developer*-nya memang jago banget. Tugas *engineer* buat estimasinya. Coding: Sulit untuk mengestimasi *velocity* untuk pertama kalinya.
K5.5	T: Terus kalau masalah *meeting*, banyak ikut *meeting* yang bukan *meeting*nya Scrum? N: Pasti ada-lah. Coding: Ada *meeting* diluar Scrum.
K5.6	T: Mengganggu ga? N: *Sometimes* ganggu. Kalau kebanyakan *meeting* jadi ga *productive* dan fokus. Kayak ada *interview*, atau gimana jadinya baru mau ngerjain, tapinya sudah di panggil lagi. Tapi kita bisa pilih kalau untuk skip *meeting*-nya. Coding: *Meeting* diluar Scrum mengganggu, terlalu banyak *meeting* mengurangi produktivitas *developer*.

K5.7	T: Sprint retronya jalan ga? Berguna? N: Berguna buat curhat. Apa beban kita. Berguna sih. Coding: Sprint Retrospective bermanfaat.
K5.8	T: Rutin dijalankan? N: Masih, tapi ga rutin lagi. Coding: Sprint retorspektif tidak rutin dilaksanakan.
K5.9	T: Ada *impact* ga dengan semakin jarang retro? N: *impact*-nya, uneg-unegnya ga bisa keluar saja. Coding: Sprint Retrospective yang tidak berjalan akan membuat keluh kesah *developer* tidak dapat tersampaikan.
K5.10	S: Pernah ada Slack ga antar *developer*? N: Dulu ada. Misalnya kita sudah *deal* dengan metode A, tapi dia *come up* dengan metode B. Jadi kesan kita, dia mau gampangnya saja. Kita mau gunakan metode A yang lebih kompleks tapi kedepannya baik. Kalau dia bisa jelaskan metode B baik, ya oke. Coding: Terjadi perdebatan antar *developer* mengenai metode yang digunakan.
K5.11	T: Cara mengatasinya? N: Kita satu tim, jadi kita *voting* saja. Coding: Voting untuk menentukan metode.
K5.12	T: Risiko pertama, tujuan proyek kurang jelas. Gimana? N: Selama ini sih, kita mau buat proyek A, sudah dikasih tahu sih sudah jelas. Coding: Tujuan proyek jelas.
K5.13	T: Kalau *requirement* dan PBI-nya? N: Sudah jelas sih. Sudah lengkap. Coding: PBI sudah jelas.
K5.14	T: Sudah jelas? Atau perlu detail? N: Requirement sih sudah lengkap. Tapi kalau mau detail, kita bahas saat bikin *task*. *User* kan mau ini itu, *how to* nya kita yang tentuin.

	Coding: Kebutuhan sudah lengkap, Melakukan pembahasan detail pada pembuatan *task*.
K5.15	T: Kalau prioritas pernah salah? N: Biasanya sih PO-nya sudah ngasih list prioritasnya. Kalau menurut kita harus diubah, bisa ngomong langsung. Selama ini sih oke-oke saja. Karena pas lagi estimasi, kita bisa *sounding*. Misalnya kita sudah buat *list*, tapi kayaknya yang ini harus dibuat dulu baru bisa jalan. Coding: Tidak terjadi salah prioritas PBI, Prioritas PBI bisa diubah dengan menyampaikan pada PO.
K5.16	T: Kalau perubahan *requirement* pernah? N: Diselipin sih *story* di tengah Sprint. Walaupun itu kerjaan gue, tapi ya tiba-tiba PO mau ada *story* ini nih. Jadinya yang harusnya bisa selesai Sprint itu malah jadi mundur ke *next*Sprint. Coding: Perubahan kebutuhan dimasukan kedalam Sprint.
K5.17	T: Penyebabnya adalah PO nya? N: Iya. Kalau PO-nya bos sendiri, ya mau gimana lagi? Coding: Perubahan kebutuhan disebabkan oleh permintaan PO.
K5.18	T: Sering? N: Akhir-akhir ini. Soalnya baru kepikiran mau ada *feature* gini. Coding: Sering terjadi perubahan kebutuhan.
K5.19	T: Cara mengatasinya bagaimana? Apa dikerjain saja? N: Kerjain saja. Saya dibayar untuk itu. Coding: -
K5.20	T: *Technical Debt*nya? N: Maksudnya yang kita kerjain tapi masih jelek di *architecture*-nya? Pernah. Coding: Pernah melakukan Technical Debt.
K5.21	T: Ada manfaatnya ga? N: Ada, Kedepannya biar *maintenancecode*-nya. Itu

	kan sebenarnya hutang yang harus diberesinkan. Coding: *Technical Debt* sebagai upaya *maintenancecode*.
K5.22	T: Pair Programming, pernah ga ada masalah ketidakcocokan. N: Kayaknya sih selama ini cocok-cocok saja. Coding: Tidak ada masalah kecocokan pada saat Pair Programming.
K5.23	T: Kalau masalah *testing*, ada *unit testing* ga? Perlu dokumen *testing* ga? N: Pernah. Kalau dari sisi gue sih yang kepikiran dari otak gue mau test apa saja. Dokumen ga sampai buat dokumen sendiri. Paling penamaan *function* untuk testnya harus jelas. Karena bagi gue mendokumentasikan sesuatu itu tugas yang melelahkan. Disini dituntut supaya bikin *method*-nya jelas, *testcase*-nya. Coding: *Unittesting* dilakukan oleh *developer*, dokumentasi dianggap melelahkan.
K5.24	T: Ada dampaknya ga sih kalau ga ada dokumentasi? N: Paling risikonya cuma *something* yang mau kamu *test*. Tapi bisa diminimalisir dengan TDD (Test Driven Development). Jadi buat *testcase* dulu sebelum *develop* something. Jadi kalau dariiven by *testcase*, chance *bug*-nya lebih kecil. Tapi ga semua nyaman pakai TDD. Tapi kita sekarang sudah mulai pakai *testcase*. Coding: Meminimalisir dampak tidak adanya dokumentasi dengan TDD.
K5.25	T: Kalau ngommongin *standupmeeting*, sudah efektif? N: Sudah efektif, 15 menit. Coding: Standup *meeting* sudah efektif.
K5.26	T: Proses Scrum di masing-masing timnya beda atau sama? N: *Basically* sih sama. Soalnya diajarin dari satu orang. Coding: Proses Scrum di tiap tim sama.
K5.27	T: Definition of done nya apa di kurio? N: Dari *engineer*, kita sudah done kalau sudah

	review, *merge* dan di-*demo*-in. Lalu, PO nya sudah setuju. Coding: Ada *definition of done* yang jelas.
K5.28	T: Kalau *velocity* rendah di awal? N: Biasanya pengaruhnya, *velocity*Sprint sebelumnya pengaruh ke Sprint selanjutnya. Dan kalau misalnya rendah, mungkin karena salah estimasi. Story yang kita anggap mudah ternyata sulit atau bisa saja salah estimasi, ternyata gampang. Tapi penting sih, supaya ada gambaran. Coding: Kesalahaan estimasi dalam menentukan *velocity*.
K5.29	T: Sering ga terjadi? N: Pernah tapi ga sering. Coding: Pernah terjadi kesalahan estimasi.
K5.30	T: Semakin banyak anggota tim di Scrum ngaruh ga dengan proses *development*? N: Kalau *development*team-nya banyak, ya cepat selesainya. Tapi kalau *developer*-nya ga jago malah membebani. Coding: Semakin banyak anggota tim yang ahli maka pekerjaan akan cepat selesai.
K5.31	T: Di kurio kan ga ada BA, perlu ga? N: Kalau BA itu dia ngerti *requirement* secara bisnis, dan *engineering*-nya. Kalau domain spesifik kayak *app* keuangan. Kita butuh *expert* di *finance*, dan kalau PO nya perlu bantuan, mungkin perlu. Tapi kurio sih belum. Coding: Belum membutuhkan bantuan Business Analyst.
K5.32	T: Komunikasi nya gimana? N: Kita komunikasi pakai Slack. Bagus-bagus saja komunikasi nya. Coding: Komunikasi dalam Scrum baik, Komunikasi dibantu oleh teknologi Slack.
K5.33	T: Perlu dokumentasi Product ga? N: Kita selalu buat dokumentasi untuk API baru. Itu sudah masuk *task* di *story*. Memang buat dokumentasi itu malas, tapi harus dibuat.

	Coding: Ada dokumentasi untuk API.
K5.34	T: Disini kan ada beberapa proses Scrum, koordinasi nya gimana? N: Misalnya supaya kita ga hambat tim *mobile* untuk *developsomething*, kita sediain dokumentasinya dulu. Itu belum jadi tapi sudah tahu ekspektasi *result*-nya gimana. Coding: Menyediakan dokumentasi API terlebih dahulu untuk digunakan oleh tim yang memerlukan.
K5.35	T: Kalau kolaborasi QA dan Pengembang? N: Komunikasinya oke-oke saja sih Coding: Tidak ada masalah kolaborasi QA dan *developer*.
K5.36	T: PO-nya sudah handal belum?? N: Sudah sih. Coding: PO sudah handal.

TRANSKRIP WAWANCARA SKRIPSI
PT TOKOPEDIA

No	**Nama: Christian Antonius** **Jabatan: Quality Assurance** **Scrum Role: Development Team**
T1.1	T: Risiko menggunakan Scrum? N: Dalam 1 Sprint, ada *task* yang sudah di estimasi 1 atau 2 minggu. Ternyata lebih, jadinya buat *task* lagi yang sama diSprint berikutnya. Lalu, kalau sudah tahu *task*nya apa saja dalam 1 Sprint, lalu ada tambahan *task* mendadak. Misal *developer*-nya sakit dan *task* jadi terbengkalai. Coding: Task tidak selesai, *task* yang tidak selesai dilanjutkan pada Sprint selanjutnya
T1.2	S: Jadi itu di*next* ke Sprint berikutnya T: Apa penyebabnya? N: Faktornya banyak sih untuk *task* ga kelar. Misalnya *developer*-nya sakit, atau ada *task* lain yang mendadak dan harus dikerjakan lebih dahulu. Masuk di tengah-tengah Sprint. Jadi *story* baru. Coding: Pengembang sakit, Ada *task* lain yang prioritasnya lebih tinggi
T1.3	T: Dampaknya?

	N: Molor ke *next*Sprint. Coding: Dilanjutkan ke Sprint selanjutnya.
T1.4	T: Sering ga sih terjadi? N: Ga semua tim kayak gitu. Kalau tim saya kan tim payment. Jadi kan tim *payment* ini suka ada promo yang mendadak. Tim saya pengaruh banget. Coding: Task yang tidak selesai dipengaruhi oleh proses bisnis dari suatu tim.
T1.5	T: Kalau rapat-rapat selain rapat Scrum banyak ga? N: Kalau lagi mau bikin *feature* baru disajak rapat. Kadang rapatnya beda, ga termasuk Scrum. Biasanya gabung dengan tim bisnis. Coding: Rapat diluar Scrum untuk *feature* baru.
T1.6	T: Mengganggu ga sih ada rapat itu? N: Kadang iya. Kalau lagi banyak tugas kok jadi *meeting* terus. Apa lagi tim promo suka menddak. Jadi mengganggu Sprint juga. Coding: Rapat diluar Scrum mengganggu
T1.7	T: Akibatnya? N: Palingan molor. Tapi ga gitu parah sih. Paling harus kelar hari ini jadinya kelar besok. Coding: Rapat diluar Scrum membuat pekerjaan tertunda
T1.8	T: Sering? N: Ga sering. Coding: Rapat diluar Scrum jarang dilakukan.
T1.9	T: Sprint retro nya ada ga? N: Ada. Di awal Sprint itu, kita ada retro dan *planning*. Jadi sebelum *planning* ada retro dulu. Sebenarnya hitungannya sama juga kayak akhir Sprint. Coding: Ada Sprint retro di akhir Sprint.
T1.10	T: Nah, Sprint retro itu berguna ga? N: Berguna sih. Sebenarnya, karena kan kita mencari tahu apasih kendala kita Sprint kemarin. Dan bisa dikurangi di-*next*Sprint-nya.

	Coding: Sprint retro berguna.
T1.11	T: Kalau retro ga dijalanin dampaknya? N: Mungkin untuk orang yang terlalu *introvert*, malah ga bisa *sharing*. Jadi kalau *introvert*-kan dipaksa ngomong dalam *meeting*nya. Kalau ga ada kan malah ga keluar uneg-uneg nya. Coding: Retro membantu *introvert* untuk menyampaikan pendapat.
T1.12	S: Pernah ada masalah pribadi ga diantara *developer*? N: Pasti ada ya. Apa lagi QA dan Pengembang kan berlawanan. QA kan cari *bugs* nya kita. Pengembang merasa sudah benar, tapi masih ada di QA. Coding: Ada masalah pribadi antara QA dan *developer*
T1.13	S: Atasinya gimana? N: Kalau awal-awal, *developer* baru mungkin merasa bingung karena ga terbiasa degan sistem Scrum ini. Tapi lama-lama pasti jadi biasa. Coding: Penyesuaian terhadap sistem Scrum.
T1.14	T: Akibat dari Slack antar QA dan Pengembang? N: Itu tergantung Pengembang-nya sendiri. Dia kan harus menyelesaikan *bugs*-nya sendiri. Mungkin kalau 1 *developer* mengerjakan banyak *task*, jadinya ya molor, karena harus *bugfix* dulu. Coding: -
T1.15	T: Tujuan proyek kurang jelas. N: Kalau tujuan kurang jelas tergantung ada *meeting* dengan *team* lainnya. Kan yang tahu nya PO. Pernah sih ga jelas kalau PO nya masih ngebayang-bayang gitu. Karena dia ga tahu jelasnya dari tim bisnis. Coding: Tujuan proyek kurang jelas pada Scrum.
T1.16	T: Dampaknya? N: Kita sudah bikin tapi malah beda sama yang tim bisnis mau. Coding: Perbedaan hasil dengan ekspektasi tim bisnis.
T1.17	S: Berarti itu mereka kurang koordinasi?

	N: Iya, kadang-kadang dibikin *meeting* semua jadinya. Coding: Kurang koordinasi antar tim.
T1.18	T: Sering? N: Ga terlalu sering sih, awal-awal doang. Coding: Kurang koordinasi terjadi di awal-awal penggunaan Scrum.
T1.19	T: Risiko *requirement* ga jelas. PBI nya kurang jelas? N: Kalau masalah itu balik lagi ke PO. Lebih sering sih jelas. Coding: PBI yang dibuat PO sudah jelas.
T1.20	T: PO nya berapa? N: 7 orang. Coding: Ada beberapa PO dalam satu perusahaan
T1.21	T: Pernah terjadi konflik *requirement* antar PO? N: PO pegang modul masing-masing, jadi ga bakalan bentrok. Kayak ada PO *payment*, Tiket, Pulsa. Coding: Pencegahan konflik kebutuhan dengan membuat modul berbeda untuk masing-masing PO.
T1.22	T: Kalau masalah prioritas kurang memadai. Misalnya ada *dependency*. N: Sering sih terjadi. Coding: Sering terjadi *dependency*
T1.23	T: Kenapa bisa terjadi? N: Kalau di tim saya, kita lagi buat fitur terganggu sama itu, kan serba mendadak. Coding: Perubahan yang mendadak menyebabkan *dependency*.
T1.24	S: Cara atasinya? N: Kalau di tim saya kan *developer* banyak. Jadi kita bagi-bagi tugas. Kita lihat *developer* mana yang ga megang waktu banyak, itu kita kasih *task* lagi. Coding: Membagi tugas *developer* yang tidak memiliki *task* banyak untuk menyelesaikan permasalahan

	depdency.
T1.25	T: Dampaknya, mereka harus ngerjain *task* tambahan. Selanjutnya, perubahan *requirement*? N: Sering sih. Penyebabnya permintaan tim. Pas lagi kerjain mereka minta ganti ya ganti. Kadang-kadang ga harus kerjain ulang sih, cuma di ubah saja. Coding: Permintaan tim untuk mengubah kebutuhan.
T1.26	T: Untuk sebagai QA kan tujuan nya *test*, ada dokumen nya ga? N: Ada setiap kali kita *test*, pasti harus ada *testcase* buat panduan. Coding: *Testcase* sebagai dokumen *testing*.
T1.27	T: Daily *standup*, efektif ga? N: Tergantung kayak ada *update* penting ga? Kalau ga ada *update*, buat apa ada *meeting*. Paling ditanya seberapa *progress* kerjaannya? Coding: Daily Scrum kurang efektif bila tidak ada update progress.
T1.28	T: Pernah ga malah bahas teknis nya? N: Kebetulan, saya kan juga Scrum Master, jadi kalau sudah melenceng, saya batasi di luar *meeting* saja. Coding: Scrum master mencegah *developer* membahas teknis di Daily Scrum.
T1.29	T: Cara tim mengerjakan Scrum nya sama ga? N: Di tokped sih beda. Menyesuaikan timnya. Coding: Proses Scrum menyesuaikan dengan tim yang bersangkutan.
T1.30	T: Definition of Done? N: Done itu, saat QA merasa yakin kalau itu siap di *release*. Setelah di-*test*, konview, *done*. IT Lead dari setiap tim akan *reviewcode* dari *developer* lain, kalau dia sudah oke dan QA sudah oke maka *done*. Coding: Ada *definition of done* yang jelas.
T1.31	T: Ada *velocity* ga di QA? N: Kayaknya *developer* saja yang pakai *velocity*, saya sih belum pernah pakai.

	Coding: QA tidak mengukur *velocity*.
T1.32	T: Efektif mana, Scrum dengan anggota banyak atau sedikit? N: Sebenarnya tergantung ruang lingkup kerjaannya, tapi kalau disini ruang lingkupnya ga terlalu besar. Jadi menurut saya lebih efektif yang sedikit karena *manage*-nya jadi lebih gampang. Saya satu tim 6 orang. Coding: Lebih mudah mengatur anggota tim dengan jumlah sedikit
T1.33	T: Masalah komunikasi, komunikasinya sudah oke? Atau ada masalah komunikasi? N: Tim saya sih oke. Misal, kalau *developer* melakukan perubahan kecil, maka dia langsung bilang ke QA. Coding: Tidak ada masalah komunikasi, Inisiatif *developer* untuk mengkonfirmasi perubahan
T1.34	T: Masalah *skill* komunikasi? Kan ada *introvert* dan *extrovert*, ngaruh ga sih ke *development*? N: Kalau di tim saya sih ga ada. Kalau pendiam jadi jarang ngomong kan? Coding: Tidak ada masalah *skill* komunikasi.
T1.35	T: Untuk setiap tim Scrum, mengerjakan dari Product Backlog berbeda-beda. Berarti ga ada kemungkinan overlap *requirement*? N: Ga mungkin. Coding: Tidak memungkinkan terjadinya overlap kebutuhan.
T1.36	T: Test-nya gimana ya? Kolaborasi antar QA dan Pengembang? N: Kalau setiap *developer* sudah merasa siap di-*test*, ya sudah di-*test*. Biasanya apa yang mereka selesain di-*test*. Coding: Test dilakukan saat *developer* sudah selesai.
T1.37	T: Kalau PO di tokped sudah handal belum? N: Pasti jelas sudah kayak gitu. Sudah tahu mana yang harus di kerjakan dulu.

	Coding: PO sudah handal.

No	**Nama: Kenneth Vincent** **Jabatan: IOS Pengembang** **Scrum Role: Development Team**
T2.1	T: Risiko di Scrum? N: Kadang-kadang kita Sprint-nya suka mundur-mundur begitu karena hal-hal gitu. Selain itu, misalnya kadang-kadang ada pekerjaan di luar Sprint, mendadak dan kita ga *plan* sebelumnya. Tapi harus dikerjakan. Coding: Story pada Sprint sering mundur ke Sprint selanjutnya, sering ada tambahan pekerjaan diluar Sprint
T2.2	T: Apa penyebabnya? N: Misalnya kita ada satu hal dari *internal*. Kita kerja sama dengan *squad* lain, tapi prioritasnya lebih rendah. Kerjaan kita malah gantung sama mereka. Coding: Dependency antar tim
T2.3	S: Cara atasinya? N: Saya kadang-kadang ada 2 kerjaan. Kerjaan pertama kita tanya tim lain. Kalau belum bisa kerjain, ya kerjain seadanya. Kan di IOS kan ada yang buat API, dan gua paling buat *design* dulu nungguin API dari mereka. Coding: Inisiatif *developer* untuk mengerjakan apa yang bisa dikerjakan selama terjadi *dependency*.
T2.4	T: Sering ga? N: Kalau gue baru sekali kejadian, dari oktober gue kerja. Coding: Jarang terjadi *dependency*.
T2.5	T: Terus, kalau masalah kerjaan yang un*plan*, kok bisa ada kerjaan un*plan* yang masuk? N: Itu kadang-kadang dari atas. Misalnya kita ada sesuatu di *app*-nya darurat banget. Ya sudah, kerjaannya jadi saya ambil alih. Dan ini dampaknya sama, kerjaan Sprint ini dilanjutin *next*Sprint. Coding: Pekerjaan yang tak terduga datang dari pihak manajemen.
T2.6	T: Seberapa sering ada kerjaan luar yang masuk yang

	un*plan*? N: Baru sekali saja sih kejadian. Tapi kerjaannya bisa di-*over* ke yang lain. Biasanya itu di-*handle* sama *squadlead*-nya. Coding: Jarang terjadi tambahan pekerjaan yang tidak terduga.
T2.7	T: Kalau Sprint retro-nya gimana? lancar ga? Berguna ga? N: Kita coba kayak dibagi 4, apa yang kita senang, ga senang, *reward* apa, dan ide untuk kedepan. Tapi kita agak jarang juga. Itu kan butuh banyak waktu. Sementara kita harus cepat. Jadi retro ga selalu saat Sprint Planning. Coding: Retro tidak selalu dilaksanakan.
T2.8	T: Tapi itu berguna, *Retrospective*nya? N: Berguna, tapi ga harus selalu sih. Mungkin 2 hingga 3 kali Sprint. Coding: Sprint Retrospective bermanfaat, Sprint Retrospective tidak rutin dijalankan.
T2.9	T: Tapi ada dampaknya ga kalau ga ada retro? N: Biasa-biasa saja sih. Coding: Tidak ada dampak signifikan dengan tidak diadakannya *Retrospective*.
T2.10	S: Pernah ada masalah pribadi ga antar *developer*? N: Kita hampir ga ada sih kalau masalah pribadi. Coding: Tidak ada masalah pribadi didalam Scrum.
T2.11	T: Tujuan proyek kurang jelas. N: Kalau gue cukup jelas sih. Coding: Tujuan proyek cukup jelas.
T2.12	T: Kalau *requirement* atau Product Backlog-nya jelas? N: Kalau itu ya kita kan dari tim bikin produk. Kita cuma eksekutornya. Kadang-kadang kita ga jelas mungkin karena kita dari sana kurang informasinya. Coding: Kebutuhan kadang-kadang kurang jelas.
T2.13	T: Dampaknya?

	N: Kita akhirnya harus diskusi sama mereka lagi sih. Terus desainnya kan sesuai sama mereka dan jadinya aneh. Jadi disesuain sama IOS-nya sendiri. Coding: Diskusi tambahan untuk memperjelas kebutuhan.
T2.14	T: Lalu, konflik *requirement* antar Product Owner? N: Itu biasanya kita ada *delegatesquadlead* yang bahas mau *requirement* apa. Coding: Mengutus *teamlead* untuk melakukan *meeting* agar tidak terjadi konflik kebutuhan antar tim.
T2.15	T: Prioritas *requirement* ga memadai? N: *Depedency* sih kejadian. Coding: Terjadi *dependency*
T2.16	T: Kok bisa? N: *Depedency*-nya itu, misalnya saya ada *task* A. Dibagi jadi A1 dan A2. A1 untuk X, A2 untuk Y. Ya yang kerjain A2 harus tunggu A1 selesai dulu. Coding: *Dependency* terjadi ketika ada *task* yang dipecah menjadi lebih kecil.
T2.17	T: Dan itu baru terjadi sekali? N: Pengalaman gue sih baru sekali. Coding: Jarang terjadi *taskdependency*.
T2.18	T: Lalu, masalah *requirement*nya berubah, pernah ga? N: Belum pernah. Coding: Belum terjadi perubahan kebutuhan.
T2.19	T: Tahu istilah Technical Debt? N: Belum. Coding: -
T2.20	T: Sesi khusus untuk Technical Debt? N: Itu kalau ada saja langsung *sharing* gitu. Coding: Sharing sebagai pengganti techical debt.
T2.21	T: Ketidakcocokan Pair Programming? N: Paling lebih ngertiin kenapa dia maunya pakai *flow* yang kayak gini. Mungkin belum nyambung.

	Coding: Terjadi ketidakcocokan Pair Programming, mencoba untuk lebih mengerti teman Pair Programming.
T2.22	S: Jadi cara mengatasinya? N: Kalau metodenya masuk akal, ya diterima. Coding: Mencoba untuk melakukan diskusi.
T2.23	T: Sering ga? N: Jarang. Coding: Jarang terjadi ketidakcocokan Pair Programming.
T2.24	T: Unit *testing* dev ga? N: Gue sih kadang-kadang suka ngetest-ngetest dulu sih kalau cukup waktu, baru kasih ke QA. Coding: Pengembang melakukan *unit testing*.
T2.25	T: Perlu ada panduan test buat *developer* ga? N: Gue sih ga perlu. Kan kita sudah tahu *flow*-nya. Coding: Tidak memerlukan panduan test *developer*.
T2.26	T: Stand up *meeting*, sudah efektif? N: Sudah oke sih. Coding: Daily Scrum sudah efektif.
T2.27	T: Berapa lama waktu nya? N: 15 sampai 20 menit. Coding: Waktu Daily Scrum sudah baik.
T2.28	T: Pernah sampai bahas teknis ga? N: Pernah, biasanya kadang-kadang yang darurat sih. Coding: Membahas masalah teknis pada Daily Scrum.
T2.29	T: Ada *impact* ga kayak gitu? N: Kalau kita dapat masukan dari *developer* lain sih. Solusi nya gini-gini. Coding: Daily Scrum dapat membantu *developer* lain untuk saling memberi masukan.
T2.30	T: Sering ga sih? N: Ga sering.

	Coding: Jarang membahas teknis di Daily Scrum.
T2.31	T: Terus, kakak selalu di tim itu ya? Belum pernah change tim. N: Iya. Coding: -
T2.32	T: Definition of done? N: Harus lolos QA. Coding: Ada *definition of done*.
T2.33	T: Kemudian, kalau masalah *velocity*? N: Enggak diukur *velocity*. Coding: Pengembang tidak mengukur *velocity*.
T2.34	T: Lebih nyaman kerja di tim Scrum yang banyak atau sedikit? N: Lebih sedikit. Kita 6 *developer* dan 4 QA. Coding: Lebih efektif bekerja dengan anggota tim Scrum yang sedikit.
T2.35	S: Termasuk banyak ya. N: Sebelumnya sih 8 baru nambah 2 orang. Coding: -
T2.36	T: Terus ada dampak ga nambahin 2 orang? N: Yang baru itu kan QA, jadi *developer* sih ga ada masalah. Coding: -
T2.37	T: Kalau komunikasi ga ad masalah? N: ga ada. Coding: Tidak ada masalah komunikasi.
T2.38	T: Kalau *skill* komunikasi? N: Kayaknya enggak sih. Coding: *Skill* komunikasi anggota tim sudah baik.
T2.39	T: Lalu kalau masalah dokumentasi *product*? N: kalau di *squad* gue sih ga ada. Coding: Tidak terdapat dokumentasi produk.
T2.40	T: Perlu ga dokumentasi *product*?

	N: Perlu sih sebaiknya. Coding: Perlu adanya dokumentasi produk.
T2.41	T: Terus masalah *dependency*, ada kan? Ada ga *product* back log yang tunggu-tungguan? N: kalau kita *mobile* lebih butuhin sih dari API. Coding: Dependency terjadi pada pembuatan API.
T2.42	T: Ada koordinasi khusus ga? N: Mungkin di Scrum lain ada prioritas masing-masing. Prioritas yang mereka rendah di kita tinggi. Jadi ya harus tunggu mereka. Coding: Perbedaan prioritas antar tim memicu *dependency*.
T2.43	T: Koordinasi QA dan Pengembang? N: Bagus banget. Kita malah deket sih. Coding: QA dan *developer* berkoordinasi dengan baik.
T2.44	T: Testingnya sih gimana? N: Kita per *feature*. Jadi kalau gue kerjain A, kalau sudah selesai test QA. Terakhir juga kita ada *test* dari awal gitu. Jadi bisa dicicil buat itu. Coding: Testing dilakukan per-*feature*, ada *testing* keseluruhan *feature*.

No	**Nama: Tri Nugraha** **Jabatan: Product Owner** **Scrum Role: Product Owner**
T3.1	T: Apa risiko penggunaan Scrum dari sudut pandang seorang PO? N: Kalau Scrum kan kita sudah ditentuin by Sprint. 1 Sprint itu 2 minggu. Karena kita pakai Scrum di perusahaan internet yang cepat banget, jadi kita sudah *plan* 2 minggu ada saja di tengah-tengah perubahannya. Terus, PO juga kadang karena perubahan datang dari *toplevel* CEO, jadi ga bisa berbuat apa-apa. Risiko kedua, Karena internet itu sangat cepat, waktu kita buat juga ga lama. Jadi harus buat *decision* yang cepat di waktu terbatas. Jadi kadang lu ga bisa *doingproperdevelopment*. Jadi kadang *designtesting* dilupakan. Terus pas sudah dibuat, malah ada *feedback* dari *user*. Jadi di balikin ke

	version sebelumnya. Coding: Perubahan kebutuhan mudah terjadi di perusahaan internet, perubahan kebutuhan datang dari pihak manajemen, harus membuat keputusan yang cepat di waktu yang terbatas, tidak dapat melakukan proper *development* di waktu yang terbatas.
T3.2	S: Kalau dapat *task* baru dari CEO, cara tanggulanginya? N: Biasanya balik lagi ke tim. Jadi PO di Tokped itu jadi jembatan antara *stakeholder* dengan *development* tim. *Stakeholder* itu kita sebutnya semua yang berkepentingan yakni *management*, *user*, *internal* tim. Nanti dari *stakeholder* atau CEO mau bikin *microsite* penjualan barang di Tokped. Akhirnya ya datang ke kita. PO negoin ke *development* team. Yang harus dijelasin di awal adalah visinya. Jadi gue datang dengan *why*-nya, kenapa lu harus menunda pekerjaan itu dan kerjain pekerjaan ini. Coding: PO bertugas melakukan negosiasi kepada *development* team untuk memasukan *task* tambahan.
T3.3	T: PO itu buat nentuin PBI kan? Gimana cara tentuin prioritas PBI? N: Zaman dulu PO yang buat PBI. Cuma makin kesini, PO sudah ga punya waktu untuk buat PBI rinci. Tapi PO cuma gambaran besarnya. Kayak lu di kapal terus lu cuma nunjukin mau ke pulau itu. Lu mau lewat mana terserah. Jadi PO tentuin tujuannya, tim yang buat PBI sendiri. Kebanyakan tim sendiri sekarang yang tulis. Biasanya gue Sprint Planning itu buat *mindmap*. Gue bagi *offense* sama defense. Kan setiap kerjaan kita ga mungkin buat *feature* baru terus. Misalnya kita terkait *scalability* atau banyak error. Itu masuk ke *defense*. Kalau *offense* bikin *feature*. Kalau *defense*, biasanya ku balikin ke tim karena kan mereka yang paling tahu mana yang masih nge-*bug*. Cara nentuin *offense* buat *feature* baru, gue *important* dulu dari pada *urgent*. Bisa dilihat *important*-nya *businessvalue* dan waktu pengerjaannya. Kalau *nice to have*, kan ga semua PBI bobotnya gede. Kadang kalau kita sudah kerjain 2 XL kadang kita masukin yang S kalau masih bisa timnya. Kita bilangnya mereka itu *squad* (*developer* dan QA). Kita harus tahu tim ini bisa kuat berapa bobot nya. Coding: PBI dibuat oleh tim, harus memahami kapasitas kerja *developer*.

T3.4	T: Setiap Sprint diusahakan selalu naik? N: Gue lebih *prefer* naik turun tapi ga *extreme*. Buat jaga *phase*-nya juga kalau lu paksa banyak kadang *burnup*. Coding: Velocity diusahakan stabil.
T3.5	T: Pernah ga ada protes mengenai mana yang bagusan dikerjain dulu? N: Kalau prioritas *defense*, mereka yang lebih terlibat. Kalau *offense*, kita saling percaya saja. Mereka juga ga perlu dibebanin, yang *defense* iya, *offense* juga. Nah jadi *offense* PO saja. Coding: Pengembang terlibat dalam prioritas *storybugs*, PO lebih terlibat dalam prioritas *story* baru.
T3.6	T: Tujuan proyek kurang jelas? N: Kalau gue pasti jelasin dulu, *WHY*, *WHAT* dan *HOW*. *Why*, kenapa proyek harus dikerjakan. What, apa yang mau kita kerjain. *How*-nya itu yang bakal lu ukur dari kerjaan lu apa. Coding: Tujuan proyek jelas karena PO berusaha menjelaskan tujuan proyek.
T3.7	T: Kalau masalah *requirement*, pernah ga *developer*-nya protes *requirement*-nya ga jelas. N: Itu banyak di awal. Yang mengalami itu biasanya yang suka bikin promo. Jadi kan promo datang tiba-tiba, *developer* nganggep *requirement*-nya kurang. Bahkan H-1 rule-nya ada lagi *rules* promo. Akhirnya kita buat satu *setrules* promo. Jadi boundaries nya gini ya. Jadi cuma bisa 1 akun handphone. Dan cuma bisa di *scope* jakarta. Diluar *scope-scopetemplate* itu, kita harus punya waktu *spend* 1 Sprint (2 minggu). kalau mau cepat harus ikuti rules-nya. Coding: Perubahan kebutuhan terjadi di awal, Membuat rule untuk perubahan kebutuhan.
T3.8	T: Sebagai PO sendiri, PBI nya pasti beda ya? N: Kita pasti beda. Kita ada 7 PO. Dan setiap PO pegang modul masing-masing. Misal gue pegang *logistic*, *goldmerchant*. Jadi kalau dia mau dapat *feature* lebih harus bayar. *Messaging* dia nanya kirim pesan ke *seller*. Ada lagi kalau PO yang mengurusi *payment* dan transaksi. Ada lagi yang CS.

	Coding: Membuat modul masing-masing untuk tiap PO.
T3.9	T: Sudah memadai prioritasnya? N: Seiring waktu belajar sih. Kalau sudah 6 bulan menggunakan Scrum biasanya kita lebih jago pakai Scrum. Kita jadi sudah bisa prediksi kalau Sprint ini ga mungkin berubah. Karena kita belajar dekat lebaran misalnya, pasti banyak perubahan promo. Jadi kita *sparescore*-nya untuk yang tak terduga. Coding: Semakin lama menggunakan Scrum, tim akan menjadi lebih handal dalam menentukan prioritas.
T3.10	T: kalau awal-awal kakak kerja gimana? N: Kita awal-awal *overestimate*, jadi cuma bisa kerjain 27 malah ambil 35. Jadinya kebawa lagi ke *next*Sprint. Jadi kita belajar *finish* yang bisa kamu *afford*. Coding: *Overestimate* pekerjaan di awal menggunakan Scrum.
T3.11	T: Sering terjadi? N: Cukup sering. Sebagai PO dan tim kita coba minimalisir perubahan ini. Kita juga kadang kasih tahu kalau kita pakai Sprint tapi diganggu-nya brutal. Coding: Sering terjadi over*estimatestory*.
T3.12	T: Kalau *stand upmeeting*, PO Ikut ga? N: Kalau ga ada *meeting* gue ikut. Tapi kalau ada, gue delegasiin ke SM. Jadi nanti gue tanya tadi ngomongin apa. Coding: PO mengikuti proses Daily Scrum, jika PO tidak dapat hadir pada Daily Scrum, maka ia mendelegasikan tugasnya ke Scrum master.
T3.13	T: *Definition of done*? N: PO Review. Tapi balik lagi, biasanya kalau tim yang dewasa itu, kalau secara teori, benar-benar Agile. Biasanya bisa tentuin sendiri *done* atau enggak. Biasanya kan *project* ada yang kecil dan skala menengah. Kalau yang kecil, kan tim yang sudah bareng sama gua sudah tahu PO *review*-nya yang ini lolos atau enggak. Kan kita juga dibantu sama *testcase*. Selain itu kita juga dibantu sama *design*. Jadi kalau di Tokped itu ada *designphase* dan *developmentphase*. Dan mereka punya tim Scrum sendiri. Kalau di tim *design*, ada UI/UX dan

	frontend. Kalau di-*development*, software *engineer* dan QA. Coding: Definition of done jelas.
T3.14	T: Berarti *design*er ga masuk ke Scrum tim *developer*. N: Kalau *project* yang besar, mulai dari *design* dulu, dirancang dulu, *design* dulu sampai keluar jadi prototype css html. Selesai itu, baru masuk Sprint-nya *development*. Kalau sudah masuk *development*, si *design*er sama *frontend*-nya ikut *daily* buat pantau. Biasanya kalau gue percaya kalau timnya sudah dewasa, biasanya yang *reviewdesign*ernya sendiri. Kan dia yang *design*, jadi tahu *pixelbypixel*. Kalau PO kan paling oh fungsinya sudah jalan. Coding: Designer ikut Daily Scrum untuk memantau pekerjaan *developer*.
T3.15	T: Kalau masalah pembobotannya sendiri, kira-kira ngaruh ga rendah diawal bobotnya gimana? N: Mereka kalau di Tokped malah *overestimate*. Biasanya kan *developer* sering nganggep bisa gampang. Tapi pas diaplikasikan malah ada *testcase* kuranglah atau gimana. Biasanya ada yang bahkan ambil 100 poin. Lama-lama kok jadi malah nyelesain ampas-ampas. Jadi satu ga selesai lanjut ke *next*Sprint. Terus ga selesai lagi, terus lanjut lagi. Makanya kalau Sprint Planning ditekankan kalau lu bisa kerjain apa yang mau lu kerjain. Coding: Terjadi over*estimate* terhadap kerjaan.
T3.16	T: Sering? N: Awal-awal sih sering. Tapi semakin kesini semakin membaik. PO juga bisa lihat pas Sprint Planning, eh lu kerjain ini, iya bisa. Lama-lama kayaknya lu kebanyakan deh. Akhirnya kita kurangi sendiri. Kalau sudah selesai kan lu bisa tambahin ditengah-tengah Sprint. Coding: *Overestimate* terjadi di awal-awal menggunakan Scrum.
T3.17	T: Kalau masalah komunikasi PO dan *developer* gimana? Ada kendala ga? Atau ada *meeting* khusus buat bahas? N: Kalau *daily standup*, komunikasi harus intens. Atau pas jam kerja juga kita sering samper-samperan

	kalau ada masalah. Terus via Slack, *email*. Abis Sprint juga ada makan-makannya. Sebenarnya lihat Scrum cuma sekedar lu bikin *software* kerjain ya sudah itu ga asik. Kalau gue sih lebih suka tim yang becandaannya sudah kayak teman benaran. Jadi banyak yang sehabis Sprint keluar, jalan bareng, nonton. Coding: Komunikasi di Daily Scrum harus fokus, Pemanfaatan teknologi untuk komunikasi, menjaga hubungan baik tim di luar jam kerja.
T3.18	T: Kalau *skill* komunikasinya kurang atau *introvert*? N: Kita kan ada test DISC. Untuk *introvert* dominan. CEO kita pak Wiliam juga *introvert* banget. Jadi ada satu *developer* yang lebih tinggi lagi *introvert*-nya. Senyum juga kayak ga pernah lihat dia senyum dan dipasangin ke tech *leadintrovert* juga. Jadi kita masuk ke tim itu seperti masuk rumah hantu. Garing banget timnya, Jadi kita coba rolling. Jadi yang rame disana kita tarik jadi Scrum Master untuk yang *introvert*. Kan kalau Scrum Master itu buat timnya *happy*. Terserah gimana caranya. Tapi sampai sekarang orangnya masih gitu. Ngubah orang itu sangat susah. Cuma karena *squad*-nya kuat dan rame, jadi bisa nularin ke dia. Kalau bisa ambil orang yang *attractive* supaya energinya positif. Coding: Melakukan rolling anggota tim, tim yang berisiskan *introvert* akan membuat suasana kerja tidak menyenangkan.
T3.19	T: Banyak ga yang *introvert*? N: Ada beberapa tapi banyak juga yang berisik. Kalau main ke lantai 6 berisik. Kalau dulu kan yang cari yang HR. Kalau sekarang timnya sendiri yang cari. Jadi kalau timnya *sreg*, ya sudah kita ambil. Jadi kalau becandaan cocok ya terima. Jadi kemungkinan dapat orang *introvert* itu di-*tackle* di depan. Coding: Team mencari sendiri anggotanya saat reqruitment.
T3.20	T: Lalu untuk *product* yang sudah jadi, ada dokumenasinya ga? N: Di kita ada PRD. Project Requirement Document. Harusnya semua PO bikin, cuma gue salah satu yang malas bikin. Jadi PRD itu isinya kayak dokumen *why*, *what* dan *businessimpact*-nya gimana, *metrix*, *technical*-nya apa, *requirement*-nya apa saja. Ada

	yang rajin buat. Tapi menurut gue PRD ga apply Agile. Lu bayangin saja lu buat *rules*, terus karena kita internet ya mainnya jadi cepat, bentar-bentar berubah. Terus lu edit lagi. Menurut gue itu wasting *time* banget. Jadi gue cuma pakai *mindmap* saja. Isinya gimana terserah, tapi *goal*-nya tercapai. Dan itu masuk KPI dan *score* gue rendah. Gue jarang banget bikin PRD. Coding: Ada dokumentasi produk berupa PRD, ada PO yang malas membuat dokumentasi. Dokumentasi dianggap membuang waktu.
T3.21	T: Gimana cara koordinasiin 3 tim yang dipegang? N: 5 Malah, 2 tim *design*, 3 *development*. *Completelydifferent* setiap tim. Dan gue merasa ga bagus. Perlu nambah PO rasanya. Karena Scrum itu harus fokus 1 saja. 3 modul itu, jadinya ga akan ada 3 squad itu kerjain 3 proyek gede. Kalau proyek gede kan butuh banyak waktu. Gue ga bisa kerjain 3 *project* gede. Jadinya maksimal 2 saja. Dan jadinya terbengkalai. Coding: Mengkoordinir lebih dari 1 tim Scrum membuat PO tidak bekerja maksimal.

No	**Nama: Hafish Herdi** **Jabatan: Android Pengembang** **Scrum Role: Development Team**
T4.1	T: Apa risiko menggunakan Scrum? N: Kita mungkin yang belum biasa sama Scrum harus adaptasi dulu. Lalu yang biasanya kerjanya kayak harus ditentuin. Misalnya hari ini ngapain, besok ngapain. Kalau Scrum kita *selforanganize*. Yang penting tujuannya tercapai. Coding: Belum terbiasa dengan *framework*Scrum, *self-organize* dari *development* team.
T4.2	S: Kalau ada anggota baru yang belum pernah menggunakan Scrum gimana? N: Kalau disini ada yang namanya Nakama Academy. Jadi nanti ada *training* dulu. Jadi dijelasin Scrum itu apa dan teknik-tekniknya. Coding: Training untuk anggota baru mengenai Scrum.
T4.3	T: Yang nge-*train* siapa? N: Biasanya 1 tim. Ada *developer* dan Scrum Master.

	Coding: Training dilakukan oleh tim Scrum sendiri.
T4.4	T: Lalu gimana cara ngubah orang yang ga biasa *selforanganize* jadi *match* sama Scrum? N: Kalau di Scrum kan ada *weeklyreport*. Awal-awalnya pasti bakal banyak tanya, hari ini mau ngapain saja. Biasanya kita kasih gambaran apa yang mau dikerjakan. Jadi kalau dia tanya, kita balik menurutmu apa yang harus dikerjakan dari gambarannya. Coding: Memberikan gambaran kepada anggota yang belum paham supaya bisa self organize
T4.5	T: Ada *meeting* lain di luar Scrum? N: Kalau di divisi ku ga ada. Tapi divisi lain ada. Kayak *sharing* gitu kalau ada *technology* baru atau gimana. Coding: *Meeting* di luar Scrum tergantung pada tim masing-masing.
T4.6	T: Kakak dari divisi apa? N: Minnion. Itu kayak *mobiledevelopment*. Coding: -
T4.7	T: Kalau di Tokped, retro nya gimana? N: Kayak Sprint Planning. Bisa sebulan 2 kali. Kayak lihat apa yang dikerjakan dan plus minusnya apa. Coding: Retrospective tidak selalu dijalankan setiap 1 *cycle*Sprint.
T4.8	T: Rutin dikerjakan? N: Iya, minimal 1 bulan sekali. Coding: Retrospective tidak selalu dijalankan setiap 1 *cycle*Sprint.
T4.9	S: Scrum ini, pernah ga ada masalah pribadi antar *developer*, yang pengaruhi pekerjaan? N: Saat ini belum sih. Coding: Tidak ada masalah pribadi antar tim Scrum.
T4.10	T: Tujuan proyek kurang jelas? N: Tujuannya jelas. Cuma di tengah dan akhir ada perubahan prioritas. Karena ada tujuan lain yang harus dikerjakan.

	Coding: Tujuan proyek pada Scrum jelas, dapat terjadi perubahan prioritas.
T4.11	T: Dampak yang dari perubahan prioritas? N: Kerjanya jadi ke-*pending*. Yang harusnya Sprint ini kelar, tapi harus tunggu lagi. Coding: Perubahan prioritas menyebabkan pekerjaan tertunda.
T4.12	T: Sering ga? N: Ga sering. Baru sekali aku. Coding: Perubahan prioritas jarang terjadi.
T4.13	T: Sudah kerja berapa lama? N: 6 bulan. Coding: -
T4.14	S: Kalau ada perubahan itu gimana? N: Kita harus ikuti. Coding: Perubahan harus tetap diikuti.
T4.15	T: Pernah ga kalau *requirement*-nya ga jelas? N: Sampai sekarang sih belum. Coding: Kebutuhan selalu jelas.
T4.16	T: Kalau Pair Programming? Pernah? N: Pernah. Coding: Pair Programming dilakukan dalam Scrum.
T4.17	T: Pernah merasa ketidakcocokan pas Pair? N: Tidak pernah. Sebenarnya Pair Programming dikerjain sendiri-sendiri. Cuma nanti disatuin. Kalau disini sih ikutin siapa yang lebih jago untuk cara-caranya atau algoritmanya. Coding: Tidak pernah ada masalah ketidakcocokan saat Pair Programming.
T4.18	T: Disini ada *unit testing*? N: Ada. Ada QA nya. Coding: Adanya *unit testing*.
T4.19	T: Stand up *meeting*-nya sudah efektif?

	N: Efektif. Coding: Daily Scrum sudah efektif.
T4.20	T: Berapa lama waktunya? N: Tergantung jumlah timnya. Paling lama sih ½ jam. Kalau sekarangkan tim gue ada 5 orang. Coding: Waktu Daily Scrum tergantung jumlah tim.
T4.21	T: ½ Jam itu bahas yang sesuai ga? Atau sampai bahas masalah teknis? N: Ya. Coding: Daily Scrum membahas hal yang efektif.
T4.22	T: Penting ga bahas teknis di Daily Scrum? N: Kalau teknisnya penting, ya memang harus dibahas di dailyScrum supaya orang lain juga tahu. Coding: Membahas teknis yang penting pada Daily Scrum.
T4.23	T: Berarti kalau Daily Scrumnya lama tidak apa apa ya? N: Tidak apa apa. Coding: Anggota tidak masalah dengan waktu Daily Scrum yang lama.
T4.24	T: Lalu, *definition of done* di Tokped? N: Ada sih. Jadi kalau dari QA sudah *clear*, sesuai *testcase*. Coding: *Definition of done* jelas.
T4.25	T: *Testcase* itu yang buat QA nya? N: Ya. Coding: QA membuat *test case*.
T4.26	T: Lalu untuk pembobotan PB, Biasanya terjadi ga pembobotan ga bisa terlalu banyak. N: Tergantung fitur yang dikerjain. Coding: Pembobotan PBI tergantung pada fitur yang dikerjakan.
T4.27	T: Kalau dari timnya sendiri, gimana bobotnya? Atau ga konstan? N: Ga konstan. Kalau kita kerjain yang fiturnya.

	Kan kita namanya *story* point. Kalau kita kerjain yang fiturnya berat tapi *storypoint*-nya kecil, itu kita bisa protes. Coding: Pembobotan *story* tidak konstan sesuai dengan kesulitan *feature*.
T4.28	T: Kalau kakak sendiri 1 tim berapa orang? N: 5 orang. Itu tapi sebenarnya kayak 2 tim. Jadi kalau aku ada tim sendiri. Itu baru 2 orang. 5 itu gabungan. Coding: -
T4.29	T: Sudah pernah kerja di tim yang jumlah anggotanya banyak? N: Kalau menurut ku maksimal 5 sih. Kalau kebanyakan juga susah *manage*-nya. Coding: Tim Scrum tidak boleh memiliki anggota terlalu banyak, tim terlalu banyak akan susah diatur.
T4.30	T: Dan kalau masalah timnya, cara mengatasi supaya manage-nya ga susah sewaktu di tim yang besar? N: Harus sering-sering dipantau. Di daily. Coding: Tim yang besar harus di pantau saat Daily Scrum.
T4.31	T: Dan jumlah anggota tim di Scrum bergantung sama proyek nya? N: Kalau jumlahnya tetap *fix*. Coding: Jumlah anggota tim tidak bergantung pada proyek.
T4.32	T: Lalu masalah komunikasi di Scrum? N: Saat ini kita komunikasi dibantu sama Slack. Kayak WA buat kantor. Sudah sesuai sama konsep Scrum. Coding: Penggunaan teknologi untuk membantu komunikasi dalam Scrum.
T4.33	T: Ada ga anggota yang *skill* komunikasinya kurang dan menghambat? N: Ada. Itu menurut aku lumayan hambat. Karena kalau dia komunikasinya kurang, jadi ga ikuti kesepakatan dan harus dibilangi lagi.

	Coding: Kurangnya *skill* komunikasi akan menghambat pekerjaan.
T4.34	T: Itu sering terjadi, anggota yang ga ikuti kesepakatan? N: Tapi itu tergantung orangnya sih. Kalau contoh disini memang kayak gitu. Coding: Ada anggota yang sulit untuk mengikuti kesepakatan awal.
T4.35	T: Terus masalah kolaborasi antara *developer* dan QA gimana? N: Kolaborasinya lumayan bagus. Jadi QA *test* kalau kita sudah selesai kerjain. Nah kan kita ada yang namanya Github. Itu ada *branch* khusus untuk *development*. Nah kalau kita kirim kerjaan kita nanti QA sudah ada notif gitu. Coding: QA dan *developer* berkolaborasi dengan baik.
T4.36	T: Untuk proyek dari Scrum sendiri ada PO sendiri yang pegang? N: Ada. Sebenarnya, tim minionnya ad 20-an orang. Tapi dipecah-pecah. Coding: Setiap tim memiliki 1 PO yang mengurusinya.
T4.37	T: Tim-timnya pernah ngerjain sesuatu yang overlap sama tim lain? N: Belum pernah karena sudah jelas setiap tim. Tapi kalau saling tunggu pernah. Jadi, fitur baru bisa dikerjain setelah tim lain selesai. Coding: Tidak pernah terjadi overlap pekerjaan antar tim, pernah terjadi *dependency* antar tim.
T4.38	T: Product yang dihasilkan ada dokumennya? N: Ada dalam bentuk wiki. Coding: Ada dokumentasi produk dalam bentuk wiki.

No	**Nama: Ryan Handy (Designer)** **Jabatan: UX Designer** **Scrum Role: Development Team**
T5.1	T: Apa risiko menggunakan Scrum? N: Kalau kita *design*, pakai Scrum, kan ada *time*line. Jadi keburu-buru gitu. Akibatnya kita *design* jadi ga maksimal.

	Coding: Desain tidak maksimal jika menggunakan Scrum
T5.2	T: Cara atasinya gimana? N: Kalau dari tim saya, kita 1 tim *design* ikut Scrum*developer* lain. Jadi *developer*Sprint Planning 2 minggu, kita dalam Scrum ada mini Scrum lagi. Jadi kita benar-benar untuk Sprint Planning minggu ini *full research* dulu, minggu kedua baru *design*. Ini ga formal sih, kayak mini Scrum didalam Scrum. Coding: Desainer menggunakan Scrum didalam Scrum untuk dapat menjalan kan tugasnya.
T5.3	T: Kalau *meeting* diluar Scrum, ada ga? N: Ada, sering sih. Saling *meeting* informal gitu. Coding: Sering *meeting* informal di luar Scrum.
T5.4	T: Mengganggu ga? N: Enggak. Justru membantu Coding: *Meeting* informal diluar Scrum tidak mengganggu pekerjaan.
T5.5	T: Lalu, kalau di Sprint tim *design*, ada retro? N: Ada. Setiap sebelum Sprint baru, kita retro. Bahas apa masalah kita, dan apa yang harus kita *stop* dan apa yang harus kita *start*. Sama *appraisal* ke team. Coding: Tim desain memiliki proses Sprint Retrospective pada Scrumnya.
T5.6	T: Rutin? N: Rutin. Sebelum Sprint Planning. Coding: Sprint retro rutin dilakukan.
T5.7	T: Kalau ga ada retro? N: Dampaknya, akan terjadi kejadian *problem* yang sama. Coding: Retrospective digunakan untuk belajar dari kesalahan.
T5.8	S: Pernah ga ada masalah pribadi antar anggota tim? N: Sedikit sih, tapi kita semaksimal mungkin untuk menyelesaikan masalahnya. Kita lebih sama-sama memahami gimana mau lu. Kalau bisa ya ga ada masalah gitu sih.

	Coding: Terjadi permasalahan pribadi antar anggota, saling introspeksi diri.
T5.9	T: Di Tokped ada? N: Sedikit sih. Gue biasanya pendekatan saja. Kenapa gini, kenapa gitu. Komunikasi saja antar tim. Coding: Jarang terjadi permasalahan pribadi di dalam Scrum.
T5.10	T: Dan orang-orang yang susah komunikasi ini ada dampak ke *development* ga? N: Ada. Contoh, kita sudah Sprint Planning buat *design* A. Tapi dia karena *miscall*, *design* A nya jadi belum ada. Coding: *Skill* komunikasi yang kurang mempengaruhi *development*, *skill* komunikasi dapat menyebabkan *miscommunication*.
T5.11	T: Tujuan proyek kurang jelas? N: Kalau tujuan sih, karena saya UX jadi harus tahu tujuannya apa. Sprint Planning, pasti gue jelasin tujuannya apa. Coding: Tujuan proyek jelas.
T5.12	T: Requirementnya ada ga sih yang ga jelas? N: Ada. Itu gara-gara komunikasi saja sih. Jadi kadang-kadang ada PO yang nambahin *requirement* tapi belum *discuss* sama kita. Coding: Kebutuhan kurang jelas karena kurangnya komunikasi.
T5.13	T: Dampaknya? N: Kita jadi ga maksimal. Coding: Kurang jelasnya kebutuhan menyebabkan pekerjaan tidak maksimal.
T5.14	S: Gimana cara mengatasinya? N: Biasanya kalau *requirement* yang kecil, kita bisa selipin ke Sprint. Tapi kalau gede kita ke *next*Sprint. Coding: Menyisipkan kebutuhan tambahan yang kecil ke dalam Sprint, mengerjakan requiremen tambahan yang besar di Sprint selanjutnya.

T5.15	T: Sering terjadi? N: Kalau di tim saya sih jarang. Karena saya menganggap, jadi harus terstruktur. Kan *release*-nya benar-benar harus teratur. Coding: Jarang terjadi perubahan kebutuhan.
T5.16	T: Dan tim Scrum kakak di pegang 1 PO? N: Beberapa. Coding: Tim Scrum desain diurus oleh beberapa PO.
T5.17	T: Ada konflik *requirement* antar PO? N: Ga ad sih konflik. Karena PO di tim gue lebih ke modul sih. Jadi PO 1 Modul Logistik. Modul 1 nya lain *topapps*. Jadi memang beda. Sekarang belum ada PO yang berdedikasi khusus untuk itu, tapi kedepannya ada. Coding: Tidak ada konflik kebutuhan antar PO, pembagian tugas PO berdasarkan modul.
T5.18	T: Dampak dari ga ad PO yang berdedikasi khusus disitu? N: Dampaknya prioritas sih. Mau duluan PO A atau B. Coding: Tim susah menentukan prioritas untuk mengerjakan permintaan dari PO.
T5.19	T: Ada ga terjadi, *dependency* di tim *design*? N: Di desain sih jarang kasus kayak gitu. Kan *design* ya *design*, ga ad tunggu-tungguan. Coding: Jarang terjadi *dependency* di tim desain.
T5.20	T: Kalau *requirement*-nya berubah gimana? N: Ada sih sering. Biasanya kita belajar dari proses. Di tengah Sprint, tim *backend* ga sanggup, jadinya *requirement*-nya diubah sama PO biar mudah dicapai. Biasanya kalau ga mateng *planning*-nya kan ekspektasinya tinggi. Coding: Sering terjadi perubahan kebutuhan, Planning yang kurang matang menyebabkan ekspektasi yang tinggi.
T5.21	T: Kalau masalah *daily standup*, sudah efektif? N: Sudah efektif. Karena setiap hari kita saling share.

	Coding: Daily Scrum sudah efektif.
T5.22	T: Penah nyerempet masalah teknis? N: Iya. Coding: Daily Scrum digunakan untuk membahas masalah teknis.
T5.23	T: Makan waktu dong? N: Bukan makan waktu sih. Jadi kita lebih tahu oh ada masalah. Jadi kita lebih *prepare* buat hadapin itu. Coding: Daily Scrum yang lama akan membuat tim menjadi tahu akan masalah dan siap menghadapi masalah.
T5.24	T: Waktu rata-rata *daily standup*? N: 20 menit sih kalau lancar. Coding: Waktu Daily Scrum efektif.
T5.25	S: Kalau ada tamabahan waktu tidak apa-apa? N: Tidak apa-apa. Kita mau selesain dulu baru kita mulai kerjaan. Coding: Penambahan waktu pada Daily Scrum tidak menjadi masalah.
T5.26	T: Kalau di tim *design*, harus ada Scrum *board*-kan? Ada *definition of done* ga? N: Kalau *design*, done itu kalau ga ada revisi lagi. Sebelum *done* kita ada in *review*. Itu kita kasih ke *developer*. Kalau sudah dikerjain, sudah naik baru diletakan di *done*. Coding: Ada *definition of done* yang jelas.
T5.27	T: Kalau untuk tim *design*, ada bobot pengerjaan setiap Sprint ga? Untuk PBI? N: Itu kita tergantung kesepakatan tim. Misal modul A kita pengen cepat. Pengen banget dinaikin, ya bobotnya gede. Itu yang tentuin kita sendiri sih. Coding: Penentuan bobot dilakukan berdasarkan kesepakatan tim.
T5.28	T: 1 tim Scrum berapa orang? N: 1 tim 5 orang, 1 nya 12 orang. Coding: -

T5.29	T: Mana yang lebih efektif? N: Yang 5 orang. Menurut saya semakin sedikit semakin efektif. Coding: Semakin sedikit anggota tim maka *development* akan semakin efektif.
T5.30	T: Dampak kalau yang banyak anggota nya? N: Mungkin makin susah *manage*-nya sih. Jadi kalau semakin sedikit kan kita gampang buat lihat di *board*-nya jelas. Kalau banyak kan semakin kacau. Coding: Susah untuk mengatur pembagian kerja pada anggota tim yang banyak.
T5.31	T: Ada proses pemecahan ga sih kalau anggotanya sudah kebanyakan? N: Kalau di tim lain (android) sih ada. Banyak orang dipecah jadi tim kecil-kecil. Coding: Pemecahan tim menjadi lebih kecil ketika sudah terlalu banyak anggotanya.
T5.32	T: Perlu dokumen *design*? N: Ada. Jadi biar kalau kita *design* konsisten. Jadi kita kerjain apa. Kayak *reportweekly*. Coding: Ada dokumen untuk desain dalam bentuk weekly report.
T5.33	T: Untuk tim *design*, keseluruhan berapa orang? N: Hampir 30. Coding: -
T5.34	T: Itu tim yang urusin urusan yang berbeda? Ga ada *overlap*? N: Ya. Coding: Tidak ada overlap kebutuhan dalam tim desain.
T5.35	T: Kalau in *review* itu, kolaborasinya gimana sih? N: Jadi kita biasanya kan Sprint kita sama *developer* barengan. Jadi kita misal kasih *design* ke *developer*, dan bisa di-*implement*. Kalau ada revisi kita ga kasih ke *done*. Kita dibalikin lagi ke *progress*. Coding: Sprint desainer dan *developer* dilakukan

	bersamaan.
T5.36	T: Dan itu ada PO sendiri yang menangani tim? N: Saya kan gabung tim apps. Tapi belum ada PO yang dedikasi khusus. Coding: Tim desain belum memiliki PO yang berdedikasi khusus.

No	**Nama: Patrict Alexander** **Jabatan: Software Engineer** **Scrum Role: Scrum Master**
T6.1	T: Apa risiko menggunakan Scrum? N: Sebenarnya, untuk bisnis *flow*-nya, mungkin lebih gampang terjadi *miscommunication*. Kalau kita ga benar-benar charge disana. Karena metode Scrum ini perkembangannya sangat cepat. Jadi misal kita sudah ada bisnis *requirement*-nya gini, tapi di tengah jalan pas lagi *develop* bisa berubah. Jadi contoh kita buat *feature* baru. Kita harus ada bisnis *flow* dulu. Kalau sudah jelas, kita ke tim *product* untuk buat *frontend*-nya. Lalu dikasih ke *developer* biasanya. Tapi ditengah-tengah misalnya dari manajemen ga mau nih kayak gini. Karena kan pakai Scrum kan ga harus sesuai spesifikasi awal kan. Ditengah-tengah harus ada *improve*ment gini-gini. Jadi *frontend*-nya harus berubah lagi. Karena bisnisnya beda. Walau *development* sudah setengah jalan. Jadi harus dirubah lagi. Jadi *developer*-nya kalau ada perubahan, ngerjainnya akan lebih lama. Coding: Mudah terjadi *miscommunication*, mudah terjadi perubahan, perubahan membuat pekerjaan menjadi lebih lama.
T6.2	T: Itu sering terjadi? N: Memang sering terjadi. Kita menggunakan metode Scrum itu karena memang kita ga mau terlalu *strict* sama yang sudah dibicarain di awal. Jadi dari tim desain dan *developer* bisa berubah dengan cepat. Coding: Perubahan sering terjadi.
T6.3	S: Cara mengatasi risiko? N: Kita ada Daily Scrum. Jadi setiap pagi kita *discuss* apa yang sudah kita kerjakan, apa yang mau kita kerjain, dan ada problem apa. Jadi untuk meminimalisir *misscom* itu menggunakan *daily standup*.

	Coding: Menggunakan Daily Scrum untuk meminimalisir *miscommunication*.
T6.4	T: Setiap perusahaan beda-bedakan penerapan Scrumnya. Beda yang di tokped sama yang diajarin ken schwaber apa? N: Biasanya di perusahaan lain, Scrum Master dekat sama internal tim dan manajemen. Kalau di tokped, SM dekat sama *developer* dan desainer. Manajemen dekatnya sama PO. Coding: Scrum master lebih dekat dengan *developer*, PO lebih dekat dengan pihak manajemen.
T6.5	T: Ada perbedaan lain? N: Belum lihat sih. Coding: -
T6.6	S: Misalkan ada anggota baru di Scrum atau gimana train-nya? N: Biasanya kalau ada anggotaa baru, kalau QA, sebagai Scrum Master kita cuma kasih arahan, suruh senior QA buat jelasin *testcase*-nya gimana, tool buat *test*-nya. Jadi biar QA baru belajar dari seniornya. Kan Scrum Master ga harus dari *developer* atau QA. Coding: Senior *developer* akan menjadi *trainer* bagi anggota baru.
T6.7	S: Misalkan dalam mengerjakan Scrum. Kalau ada *story* yang ga selesai dalam 1 Sprint? N: Biasanya Scrum kita 1 sampe 3 minggu. Biasanya kalau mepet banget, Sprint Planningnya kita kejar dalam 1 minggu. Tapi kalau lagi normal-normal saja, 2 minggu. Dan kalau lagi banyak liburan, kan sekarang mau lebaran dan minggu depan banyak yang cuti. Jadi kita perpanjang jadi 3 minggu. Di akhir Sprint kita ada Sprint Review. Itu kita tanya-tanya tim halangannya apa sih dalam sisi *developer*, atau *design*, atau QA-nya. Misalnya kita ada *environmentstaging*. Disana kita bisa ubah. Dan biasanya yang ubah bukan cuma tim kita saja. Biasanya tim lain itu ada juga yang bisa buat *bugs*. Jadi itu salah satu faktor yang bisa dikasih tahu pas Sprint Review. Kalau retro itu kita pakai *start* sama *keep*. Kalau *review* itu kita ke *story-story*nya, ga selesai kenapa, *goal*-nya apa. Jadi pas *planning* ada gambaran kedepannya gimana.

	Coding: Waktu Scrum bergantung pada kebutuhan dan kondisi.
T6.8	T: Disini ada tujuan proyek kurang jelas? N: Enggak sih. Biasanya kita di akhir Sprint Review, kita sudah tentuin *goal* kita mau ngapain *next*Sprint-nya. Coding: Tujuan proyek jelas karena sudah ditentukan setiap Sprint Review.
T6.9	T: Kalau *requirement*-nya jelas? N: Kalau itu kita biasanya bahas sertelah tentuin *goal*-nya apa. Lalu kita pecah *goal* ini siapa yang kerjain. Kemudian kita pecah jadi *task* kecil-kecil. Misalnya kita untuk kembangin halaman produk, kita kita tim mana dulu. Kalau sudah tahu *requirement*-nya, kita kembangin. Coding: Kebutuhan jelas karena dibahas setelah menentukan *goal*.
T6.10	T: PO-nya pegang modul masing-masing, jadi ga mungkin overlap? N: Ya. Coding: PO mempunyai modul masing-masing.
T6.11	T: Kalau Scrum master ikut ga proses prioritas *requirement*? N: Biasanya yang prioritasin itu PO. Kita paling cuma kasih masukan *developmenttime*nya, dan kasih pendapat kalau ada *dependencystory*. Coding: Prioritas kebutuhan dibuat oleh PO.
T6.12	T: Biasanya terjadi perubahan *requirement* karena apa? N: Ditengah-tengah pengerjaan, ada saja yang baru kepikiran. Kalau ini dapat memicu ini itu, jadi ada perubahan. Mungkin pada awal, *requirement*-nya kurang matang. Kurang sumber. Pas di tengah baru sadar. Coding: Planning kebutuhan kurang matang dan kurang sumber.
T6.13	T: Pair Programming? N: Kita disini jarang pakai Pair Programming. Jadi kalau memang ada kerjaaan bareng, kita pakai Github buat kolaborasi.

	Coding: Jarang melakukan Pair Programming, mengguakan teknologi untuk menggantikan Pair Programming.
T6.14	T: Lalu, untuk *stand upmeeting*, Scrum Master selalu terlibat? N: Daily Scrum kita sebagai Scrum Master cuma memantau. Kita mastiin *story* yang dibuat kecil-kecil itu harus ada yang gerak setiap hari. Kalau ga gerak berarti kurang kecil. Coding: Scrum master memantau proses Daily Scrum dan Sprint yang berlangsung.
T6.15	T: Untuk melakukan *meeting* setiap hari itu, harus di-*initiate* Scrum Master atau memang sudah rutinitas setiap hari? N: Yang initiate tetap Scrum Master. Jadi kita tanya anggotanya mau jam berapa. Kadang ada yang ga bisa kan. Jadi Scrum Master memastikan semua anggota dapat hadir. Kalau mereka sudah biasa, ya mereka *standup* sendiri tanpa SM. Coding: Scrum Master mengajak anggota tim untuk melakukan Daily Scrum.
T6.16	T: Kemudian, ini kan ada banyak tim, proses Scrum yang dijalan kan setiap tim sama ga? N: kalau metode Scrumnya sendiri, setiap tim sama kurang lebih. Tapi disini ada 2 sih, ada Scrum ada Kanban. Jadi ada beberapa tim yang pakai Kanban. Kalau pakai sistem Sprint, mereka ga tahu apa yang mau dikejar Sprint ini. Jadi mereka lebih enak kalau ada *task* baru dikerjain. Coding: Proses Scrum di tiap tim sama.
T6.17	T: Lalu, kan ada Scrum*board* kan. Kalau *definition of done*-nya gimana? N: Saat *feature* itu sudah live dan bebas dari *bugs*. Masing-masing tim beda-beda sih. Tergantung kebutuhan tim. Bisa saja ada yang harus buat dokumen baru di done-in. Coding: *Definitionofdone* berbeda tiap tim, *definition of done* jelas.
T6.18	T: Pada awal Sprint kan ada pemilihan PBI yang mau dikerjakan, pembobotan PBI nya gimana? N: Biasanya setelah tentuin *storypriority* kita,

	goal yang kita tentuin. Dan setelah *goal* ditentuin, kita tentuin mana *priority* 1, 2 dan 3. Dan setiap *goal* kita pecah *story*-nya masing2. Penilaian nya sih dari *developer*-nya sendiri. Misal *developer* ini in charge untuk kerjain *goal* ini, nah dia harus kira-kirain sendiri seberapa lama dia bisa kerjain *goal* itu. Coding: Pembobotan PBI dilakukan oleh *developer* yang akan mengerjakannya.
T6.19	T: Kalau semakin banyak anggota tim menurut kakak lebih bagus atau tidak? N: Kalau misalkan kerjaannya lagi banyak, masing-masing bisa dikerjain kerjaan masing-masing. Kalau kerjaan nya lagi sedikit, pembagian tugasnya jadi ga merata. Jadi ada yang nganggur-nganggur gitu. Coding: Keefektifan jumlah anggota tergantung pada kondisi proyek.
T6.20	T: Jumlah anggota setiap tim di Tokped berapa rata-rata? N: Pengembang rata-rata ada 3, QA rata-rata ada 2. Total 5 sih. Coding: -
T6.21	T: Kalau masalah komunikasi antar anggota tim Scrum? N: Miscom biasanya sudah kita antisipasi di Daily Scrum. Kita biasanya jarang banget sih, setahun 2 kali paling. Dalam menentukan *goal* itu, *requirement*-nya biasanya ini, terus jadinya yang dibuat bukan ini. Jarang banget sih. Coding: Menggunakan Daily Scrum untuk mengantisipasi *miscommunication*.
T6.22	S: Kalau miskom gitu gimana kak, mengatasinya? N: Nah itu, biasanya waktu pertama kali terjadi miscom, langsung masuk retro. Jadi kita tahu dimana miskom nya. Jadi untuk *flow* yang terjadi miskom itu di-*improve* lagi. Coding: Menggunakan *Retrospective* untuk mengatasi *miscommunication*.
T6.23	T: Kalau masalah kemampuan komunikasi setiap orang? N: Kalau di Tokped ga tahu sih ada kayak gitu atau

	enggak. Tapi itu ngaruh. Biasanya *softwareengineer* yang suka kurang bisa komunikasi. Biasanya mereka diam saja, mereka kayak anggap gue ngerti nya kayak gitu. Cuma mereka ga klarifikasi lagi sudah benar belum yang mereka ngerti. Pas di test QA baru salah. Jadi *takesdevelopmenttime* lagi. Coding: *Skill* komunikasi memperngaruhi proses *development*.
T6.24	T: Sering? N: Enggak sih. Coding: Jarang terjadi *miscommunication*.
T6.25	T: Dokumentasi *product* ada? N: Kalau di Tokped, dokumentasi kurang banget sih. Kita kan kembangin *feature* dan *improve*ment cepat banget sih. Harus ada *deadline*-nya. Tapi kita suka lupain dokumentasi. Jadi kalau misal ada orang baru, atau *feature* baru, kita harus lihat kebelakang lagi gimana sih *feature* sama *flow* kita. Coding: Kurangnya dokumentasi produk, sulit untuk mengajarkan anggota baru tanpa menggunakan dokumentasi.
T6.26	T: Dampaknya jadi lama lagi? N: Ya. Coding: Kurangnya dokumentasi menyebabkan proses pembelajaran anggota baru terhadap produk yang ada menjadi lama.
T6.27	T: Ada *dependency* ga antar tim Scrum? N: Ada. Coding: Terjadi *dependency* antar tim Scrum.
T6.28	T: Penyebabnya? N: *Dependency* itu sendiri dari *product* itu sendiri ya. Kan *product* ini misalnya, *flow* pembelian, ada tim yang ngurus *product*, ada tim yang ngurus pembayaran, ada tim yang ngurus transaksi. Jadi untuk lewati *flow* itu, misal kan kita di tengah-tengah, transaksi, kita harus bedain *product* A dan B. Tim *product*-nya harus ada *developer*an dulu buru tim transaksi bisa tampilin data tim *product*-nya. Coding: Terjadinya *dependency* disebabkan karena urutan proses bisnisnya.

T6.29	T: Cara mengatasi supaya *dependency* ga selalu terjadi? N: Dari tim *product*, PO-nya harus lebih matang untuk tentuin. Misal dia tahu 2 bulan kedepan ada *goal* ini. Dia sudah harus tahu *flow*-nya butuh tim ini-ini untuk *incharge* disana. Jadi sebelum kasih tim transaksi tadi, dia sudah harus kasih 1 bulan duluan untuk tim *product*. Coding: *Planning* yang lebih matang dari tim produk.

No	**Nama: Ryandy Law** **Jabatan: Web Pengembang** **Scrum Role: Development Team**
T7.1	T: Apa risiko penggunaan Scrum? N: 1. Kerjaan ga selesai, 2. Arah pengerjaan ga jelas, 3. Metode pengukurannya enggak sesuai atau ga jelas. Karena kalau di Scrum kan ada *scoring*-nya. Tapi *mostly* alasannya karena belum fasih menggunakan Scrum*method*. Coding: Pekerjaan terkesan seperti tidak selesai, tidak jelasnya arah pengerjaan, tidak jelasnya metode pengukuran bobot atau prioritas, belum fasih dalam menerapkan Scrum.
T7.2	T: Untuk kerjaan yang ga selesai karena apa? N: Itu karena karakter individu masing-masing. Kadang terlalu pede kerjaan dia pasti selesai. Dia lupa memperhitungkan bantuan dari sisi-sisi lainnya. Apakah karena kita *develop* software, kan *cycledevelopment*-nya berapa lama, *cycletesting*-nya berapa lama, *cyclebeta* berapa lama baru bisa dirilis. Jadi mostly gagal dalam ngecek maksudnya, ngeprekdiksi *time*-nya. Jadinya ngambil kerjaan terlalu banyak jadi ga ada yang selesai. Coding: Karakter individu mempengaruhi pekerjaan yang tidak selesai.
T7.3	T: Kalau arah ga jelas? N: Kan harus ada *role* yang namanya PO untuk nge-guide kita mau kemana. *Initial*-nya *role*-*role* itu ga terlalu di-*utilizefunction*-nya. Kayak PO kita ga perlu, ga penting, atau Scrum Master-nya gitu. Jadi *role*-*rolesignificant* yang buat nge-*guide* malah ga perlu dulu, dan jadinya *goal*-nya berantakan.

	Coding: Kurangnya kepercayaan untuk meminta bantuan *role* yang ada di Scrum.
T7.4	T: Kalau metode pengukuran ga selesai? N: Ok, anggapannya gini, lu mau kerjain semua sistem *frontend* dan *backend*. Kan punya kesulitan masing-masing. Tapi masing-masing orang ada yang bilang *frontend* yang lebih susah, satunya *backend* yang lebih susah. Tapi *mostly*Scrum itu lebih ke orangnya sendiri yang *handle* supaya berjalan dengan baik. Coding: Sulit dalam melakukan proses scoring.
T7.5	S: Perlu pakai *scoring*-nya kertas 1 – 5 gitu? N: Dulu kita kayak gitu. Tapi gue pribadi kurang suka. Terlalu panjang waktunya. Karena buang-buang waktu. Mending dimanfaatin ke *planning* atau *preparation*-nya. Karena kalau lu cuma hitung, ternyata *preparation*-nya kurang, ya pasti gagal. Coding: scoring menggunakan kertas membuang-buang waktu.
T7.6	S: Metode pengukuran yang bagus gimana? N: Di tim kita jadi sama sekali ga itung *score*. Tapi kita *do bypriority*. Kita me-*reduce* metode *scoring*. Coding: Mengerjakan sesuai prioritas tanpa melakukan *scoring*.
T7.7	T: Kalau *meeting* diluar Scrum? N: Selalu ada sih. Gue kan ada kerjasama dengan *thirdparty*. Coding: Ada *meeting* diluar Scrum.
T7.8	T: Ganggu ga? N: Sebenarnya sih ngegganggu. Apalagi saat lu lagi kerjain sesuatu. Tapi sebenarnya membantu untuk *project* kedepannya. Kalau ga di-*meeting*-in ga nyamain pandangan, akhirnya tujuannya ga ketemu. Coding: *Meeting* diluar Scrum menggaggu waktu *developer*, *meeting* di luar Scrum membantu memperjelas proyek kedepan.
T7.9	T: Salah satu *step*Scrum adalah retro. Jalan ga? N: Dulu jalan. Tapi karena terlalu banyak waktu

	yang digunakan, jadi gue hilangkan. Kebetulan di tim gue kalau ada masalah pada langsung bilang. Jadi ga perlu retro lagi. Coding: Retrospective dihilangkan karena memakan waktu.
T7.10	S: Pernah ada masalah pribadi ga di tim Scrum? N: Mungkin ada, tapi gue ga tahu. Gue sih ga ada. Kalau gue ga seneng sesuatu pasti sudah langsung gue omongin. Diskusiin lebih lanjut. Coding: Tidak ada masalah pribadi di dalam tim Scrum, melakukan diskusi informal untuk menyelesaikan masalah.
T7.11	T: Tujuan proyek kurang jelas pakai Scrum? N: Selama timnya atau semua fungsinya berjalan dengan baik, *goal*-nya pasti tercapai. Di tim gue sih sudah *autonomous* juga, jadi ga ada kayak gitu. Coding: Tujuan proyek sudah jelas.
T7.12	T: Kalau *requirement* ga jelas? N: itu pasti kejadian. Jadi pada saat *development*, kita selalu cari ini sesuai ga, ini sesuai ga. Jadi langsung direct *feedback*. Coding: Kebutuhan tidak jelas.
T7.13	T: Penyebabnya ga jelas? N: Karena yang diberi itu *generalidea*-nya. Coding: Kebutuhan tidak jelas karena hanya diberikan ide umumnya saja.
T7.14	T: Terus dikembangkan sendiri? N: Ya. Coding: Pengembang harus menentukan sendiri apa yang harus dilakukan berdasarkan *generalidea* yang diberikan.
T7.15	T: Konflik *requirement* sama PO lain? N: Selalu sih. Kan masing-masing PO punya *goal* masing-masing. Tapi kadang *goal*-nya, karena perusahaan kita sudah mulai besar, kadang perubahannya ga diberitahukan. Jadi konflik of interest pasti ada. Coding: Terjadi konflik kebutuhan antar PO.

T7.16	T: Sering ga? N: Ada pastinya. Tapi kalau sering sih ga terlalu sering. Karena kita sudah dipecah jadi per modul. Coding: Tidak terlalu sering terjadi konflik kebutuhan.
T7.17	T: Proses memprioritas*k*annya gimana? N: Dapat dari PO dulu. Goal-nya di Sprint ini apa. Jadi masing-masing orang kejar masing-masing targetnya sendiri. Coding: PO menentukan prioritas yang ingin dikerjakan.
T7.18	T: Pernah ga prioritasnya ga *reliable*? N: Kejadian kayak gitu terjadi jika kita gagal *sincron* di bagian manajemennya. Tapi di kita ga ada sih. Awal-awalnya saja yang kayak gitu. Coding: Kesalahan prioritas terjadi di awal menggunakan Scrum.
T7.19	T: Kalau perubahan *requirement*? N: Selalu itu. *Mostly* mungkin perencanaan kurang matang, dan pengetahuan terhadap *product* kurang. Coding: Selalu terjadi perubahan kebutuhan, Pernecanaan kurang matang, kurangnya pengetahuan produk.
T7.20	T: Dan dampak dari perubahan *requirement*-nya? N: Lembur. Coding: Perubahan kebutuhan membuat *developer* harus menambah waktu kerja mereka.
T7.21	T: Sering banget? N: Ya cukup sering. Coding: sering terjadi perubahan kebutuhan.
T7.22	S: Cara mengatasi perubahan *requirement*? N: Ya kerjain saja. Kita kan ceritanya masih *startup*. Jadi sudah biasa. Coding: Perubahan *requirement* harus tetap dikerjakan.
T7.23	T: Kalau Pair Programming pernah?

	N: *Currently* sih enggak. Kita sih menghargai privasi. Jadi kalau Pair Programming kan orang bisa lihat, kalau gue pribadi sih kurang. Coding: Tidak melakukan Pair Programming, Pair Programming dianggap kurang menghargai privasi.
T7.24	T: Apakah ada *unit testingdeveloper*? N: Kalau untuk itu sih selalu ada. Cuma efektif atau enggaknya biasanya lebih efektif QA. Kadang kita sudah bikin sistemnya dan *goal*nya seperti itu. Kalau *developer* sih enggak kepikiran cari celah nya biasanya. Coding: Ada *unit testing*, *testing* lebih efektif dilakukan oleh QA
T7.25	T: Ada dokumen *testing* untuk *developer*? N: Sampai saat ini sih belum ada. Kalau di sistem baru kita sih ada, kita sudah *create* sendiri *unittest*-nya, tinggal di-*test*. Coding: Tidak ada dokumen *testing* untuk *developer*.
T7.26	T: Perlu ga dokumennya? N: Perlu, karena kalau ga ada, orang-orang yang baru jadi agak susah untuk nge-*replacepreviousprorgammer* ga bisa cepat *follow* sistemnya. Tapi kita malas buatnya. Coding: Perlu adanya dokumentasi.
T7.27	T: Stand up *meeting*? N: Efektif sih ga pernah efektif karena tim nya dari kecil *grow* terus. Selain dari mensinkron apa yang telah dikerjain. Coding: Daily Scrum sulit untuk efektif karena banyak anggotanya.
T7.28	T: Penyebabnya? N: Mungkin anggotanya terlalu banyak. Coding: Terlalu banyak anggota menyebabkan Daily Scrum semakin kurang efektif.
T7.29	T: Kalau bahasannya sesuai ga? N: Biasanya sih sesuai *up to certaintime*. Kalau sudah kelar, ada continual-nya buat bahas teknis nya.

	Coding: Hal yang dibahas pada Daily Scrum sudah sesuai.
T7.30	T: Dampak dari anggota terlalu banyak? N: Negatif nya sih cuma sedikit lebih lama saja. Coding: Daily Scrum menjadi lebih lama di tim Scrum yang jumlah anggotanya banyak.
T7.31	T: Kalau rata-rata waktu Daily Scrumnya? N: Setengah jam sih gue. Orangnya sudah 11 soalnya. Kita juga masukin tim luar yang masuk untuk ceritain *problem* yang terjadi. Coding: Waktu Daily Scrum sudah baik.
T7.32	T: Belum ada di pecah ya? N: Sebenarnya core-nya 5 *developer*, dan 3 QA. Sebnernya belum perlu dipecah. Biasanya gara-gara tim lain juga yang bilang buat *problem* dan kita kasih solusi langsung, makanya jadi lama. Coding: -
T7.33	T: Proses Scrumnya beda? N: Ya. Beda. Ada yang sistemnya langsung pakai kanban. Ada juga Sprint itu cuma buat *task* list doang. Coding: Proses Scrum antar tim berbeda.
T7.34	T: Tetap pakai ScrumBoard kan? N: Ya. Coding: Scrum *board* digunakan sebagai media.
T7.35	T: Definition of done? N: Kita tergantung dari *initialstatement* mau ngapain. Kalau misal lu buat prototype sistem, kan ga di-*deploy*, jadi kalau sudah di-*test* ga ada *bug* ya done. Tapi kalau sistem yang *deploy*, baru done kalau sudah *deploy*. Coding: Ada *definition of done* yang jelas.
T7.36	T: Business Analyst ada? N: ada. Coding: Ada Business Analyst dalam Scrum.
T7.37	T: Business analyst itu bentrok sama PO ga?

	N: Pasti bentrok. Bisnis kan ngejual sesuatu, kalau PO nge-*launch* sistem atau *stabilize* sistem. Jadi kadang mau stabilin sistem dulu tapi tiba-tiba mau buat *product* baru dari internalnya, jadi ga bisa. Coding: Business analyst terkadang bertentangan dengan PO.
T7.38	T: Kalau dari tim kakak, komunikasinya gimana? N: Lancar saja, efektif. Coding: Komunikasi berjalan dengan baik.
T7.39	T: Kalau masalah *skill* komunikasi setiap anggota gimana? N: Enggak sih di gua. Kalau ada yang kurang sih kita langsung disajak ngobrol. Ga perlu di tutupi. Coding: Tidak ada masalah dengan *skill* komunikasi.
T7.40	T: Belum ada dokumentasi *product* akhir? N: Kalau sistem lama belum ada. Kalau baru ada. Kita kan punya 2 sistem. Coding: Sudah ada dokumentasi produk akhir.
T7.41	T: Kalau antar tim Scrum ada koordinasi? N: Perlu. Biar sudah di pecah kecil-kecil, tapi kan pasti tetap ada modul yang saling perlu koordinasi dulu. Coding: Perlu adanya koordinasi antar tim Scrum.
T7.42	T: Kalau sama QA kolaborasinya? N: ya setiap kalau sistem selesai, dicek dulu, kalau sudah oke baru *deploy*. Coding: Kolaborasi *engineer* dengan QA sudah baik.
T7.43	T: Dan itu, tim kakak, dipegang oleh 1 Product Owner khusus? N: Iya. Coding: Satu tim dipegang oleh satu Product Owner khusus.

TRANSKRIP WAWANCARA SKRIPSI
PT HappyFresh

No	**Nama: Nu'man Naufal** **Jabatan: Junior software *engineer* (frontend)** **Scrum *role*: Development Team**
H1.1	T: Apa risiko pakai Scrum? N: Kalau Scrum itu kan dari Sprint-Sprint gitu. Jadi lebih cepat, lebih setiap Sprint itu sudah tahu mau *launch*-nya gimana. Setiap Sprint itu sudah harus sudah bisa *release*. Nah itu memang jelas banget *requirement* dari tiket-tiket nya itu. Setiap hari juga bisa di-*track* hari ini kerjain apa dan ada hambatan atau enggak. Walaupun ada tiket yang besar harus di pecah-pecah. Ini kan *happy* data ya, misalnya buat barchart, itu kan kompleks, kerjaannya ga sehari selesai. Bagus sih dipecah-pecah *story*-nya. Risikonya, mungkin pengalaman ya. Dulu itu kalau Scrum, dulu pernah ada *story* yang spec-nya belum jelas. Terus kan dikerjain dengan spec yang asumsi sekarang. Nah abis itu minggu depan dia baru update *story*-nya. Dan pasti jadi berubah. Kita jadi lebih butuh waktu lama untuk kerjain ini. Harusnya kan Sprint Planning dulu, terus kick off baru jalan. Di Sprint Planning itu memang sudah harus jelas spec setiap *story*. Jangan di-*update* ditengah-tengah. Nah, itu sih yang buat terhambat. Coding: Story pada Scrum yang kurang jelas, Perubahan *story*.
H1.2	T: Berarti Sprint Planning-nya harus benar-benar jelas ya? N: Sayakan *happy* data ini memang *scratch* dari 0. Setiap Sprint kan ada retro nya, nah itu memang bantu sih evaluasi Sprint lalu apa saja yang kurang. Coding: Sprint Retrospective membantu untuk mengevaluasi Sprint yang telah dikerjakan.
H1.3	S: Kalau Sprint Planning-nya ga jelas, kakak gimana? N: Kan ada Jira itu ya, *story-story* nya di-*comment* disana. Coding: Menggunakan Scrum Board virtual untuk memperjelas *story*.
H1.4	T: Itu kayak Scrum*board* nya ya?

	N: Iya, Scrum*boardvirtual*. Kita kasih komentar di-*story* tersebut yang belum jelas. Kalau saya di *happy* data, PM-nya kan orang luar, ga disini, dia bales nya ga hari itu juga. Jadi di tengah-tengah baru di-*update*. Coding: Scrum *board* virtual untuk membantu komunikasi mengenai *story*.
H1.5	T: Kalau *meeting* selain Scrum apa? N: Ada sih sama orang bisnis. PO nya. Dia pengen share keadaan bisnis nya gimana sekarang, Coding: Ada *meeting* diluar Scrum, *meeting* dengan orang bisnis.
H1.6	T: Berguna ga? N: Berguna sih, kita kan ga cuma tahu *development*-nya doang, perlu juga ngerti bisnisnya. Kita harus terbuka satu sama lain. Coding: *Meeting* dengan pihak bisnis berguna bagi *developer*.
H1.7	T: Itu *meeting* nya mendadak ga atau pernah merasa mengganggu ga? N: Enggak sih, Cuma bentar. Ga ganggu ganggu banget. Coding: *Meeting* diluar Scrum tidak mengganggu *developer*.
H1.8	T: Ada retro kan. Lancar ga? Bermanfaat ga? N: Pertama-tama kan *lead*-nya beda. Terus april belakangan ganti yang baru, dia memang Scrum master yang lebih *strict* yang dulu itu. Tapi sekarang sudah *missed* 2 retro. Memang guna sih, kan ada *save*, *meet*, kategorinya. Terbuka jadinya. Coding: Sprint Retrospective berguna, pelaksanaan Sprint Retrospective tergantung pada Scrum master.
H1.9	T: Ada dampaknya ga sih setelah ga ada retro? N: Uneg-uneg jadi ga keluar. Kayak ini tim dulu jam 10 harus daily *stand up*, sekarang terngantung siapa yang datang terakhir. Coding: Pengembang tidak dapat menyampaikan keluh-kesah tanpa Sprint Retrospective.
H1.10	S: Kalau Scrum ini kan terkenal kerja tim. Pernah ada masalah pribadi yang dibawa ke kerjaan dan

	menghambat kerjaannya. N: Kalau yang pribadi sih enggak. Tapi kalau perbedaan pendapat mengenai arsitektur apa itu ada, dan didiskusikan. Saya kan baru *frontend* di sini, saya bingung mau pakai teknologi apa. Kita juga minta bantuan sama tim lain mau pakai apa. Dan didiskusikan Coding: Tidak ada masalah pribadi dalam Scrum, terjadi perbedaan pendapat mengenai arsitektur yang akan digunakan.
H1.11	S: Ada berapa tim Scrum? N: Setiap meja 1 tim. 4 atau 5 gitu. Coding: -
H1.12	T: Tujuan proyek kurang jelas? N: Tujuannya jelas sih. Kalau *product* dari 0 kan jelas banget, Bikin *dashboard* untuk *happy* data. Coding: Tujuan proyek jelas untuk produk yang dibangun dari awal.
H1.13	T: Kalau *requirement*-nya jelas? N: Ga detail. Bukannya ga jelas. Dan berdampak perubahan itu. Misalnya tadi ada *barchart* tapi ada nilai *growth* yang memungkinkan minus. Di awal kan sudah ditanyain mau bikin *chart*-nya gimana. Tapi dia mau tapi 0 sampe segini. Akhir-akhir ini diubah, yang *negative* gini. Coding: Kebutuhan kurang detail mengarah pada perubahan kebutuhan.
H1.14	T: Sering ga berubah karena ga detail. N: Ga sering2 banget, tapi ada lah. Coding: Jarang terjadi perubahan kebutuhan karena kurang detail.
H1.15	T: Kalau *leveling*, *low*, *medium*,*high*? N: *Medium*. Coding: Jarang terjadi perubahan kebutuhan karena kurang detail.
H1.16	T: Ada acara mengatasi nya ga kalau ga detail? N: Kita ga bisa kerjain *story*-nya sampai benar-benar jelas. Letakan di prioritas bawah saja. Kan

	capek juga kalau sudah kerjain terus berubah. Coding: Tidak bisa mengerjakan *story* yang belum jelas, mengubah prioritas *story* yang belum jelas.
H1.17	T: Berapa Product owner-nya? N: Kalau di *happy* data kan ada Scrum Master dan Product Manager dan ada yang punya *product* sendiri. Kalau PM ini yang menjembatani bisnis dengan *developer*-nya. Coding: Product Owner disebut sebagai Product Manager.
H1.18	T: Berarti ini PM. N: Saya ga tahu sih tim lain. Tapi tim saya satu. Setiap tim harusnya ada. Coding: Setiap tim diurus oleh satu Product Owner.
H1.19	T: Pernah ada konflik *requirement* ga? Atau setiap tim kerjain yang benar2 beda. Jadi ga akan ada yang overlap. N: Kalau saya sih enggak ada. Tapi ada *dependency*. Coding: Tidak ada overlap kebutuhan, terjadi *dependency*.
H1.20	T: Berarti ada *dependency* ya. Kira-kira penyebabnya apa ya? N: Mungkin dari tim lain juga sih lagi kerjain apa sekarang, diakhir ga dapat kita. Contoh nya kita buat *dashboard* buat promosi. Tapi promosi yang ada di HappyFresh belum bisa di terapkan di *dashboard* kita karena memang lagi dikerjain di tim promosi. Coding: Kurangnya koordinasi antar tim menyebabkan *dependency*.
H1.21	T: Dan dampaknya dari *dependency* ini? N: Feature itu ga sesuai sama yang diharapkan dan butuh waktu lagi untuk kerjain. Coding: Waktu untuk menyelesaikan suatu *feature* menjadi semakin lama sebagai dampak dari *dependency*.
H1.22	T: Dan ini lumayan sering ga terjadi? N: Baru satu sih. Saya kan orang baru. Coding: Jarang terjadi *dependency*.

H1.23	T: Tapi cara atasinya harus diskusi dulu. N: Ya. Coding: Dependency diatasi dengan melakukan diskusi terlebih dahulu.
H1.24	T: Kalau *requirement* yang ga jelas tadi sudah ya (ga detail). Kalau disini ada Technical Debt ga? Ini bagian dari Sprintnya atau diluar Sprint. N: Tergantung *developer*-nya itu. Harusnya sih di dalam Sprint. Kalau memang *feature* sudah selesai dan masih ada sisa hari buat Bug *fixing*. Kalau ga ada *bugs*, ya kita Technical Debt. Misalnya *developer* ga puas dengan *code* nya dan mau *refactoring* ga ada waktu lagi, ya diluar jam kerja tapi ga diwajibkan sih. Coding: *Technical Debt* dilakukan diluar Sprint.
H1.25	T: Dan ini ga rutin ya? N: Enggak. Kalau memang harus ada saja. Coding: *Technical Debt* dilakukan saat dibutuhkan.
H1.26	T: Pair Programming ada? N: Ada. Coding: Ada Pair Programming didalam Scrum.
H1.27	T: Pernah ada masalah ketidakcocokan saat Pair Programming ga? N: Di tim saya kan ada 2 *backend*, 1 *frontend*, 1 QA dan 1 *lead*-nya. 2 *backend* nya ini yang dari pertama pair dulu buat nyatuin pendapat tech-nya apa dan nyatuin stadarnya. Ini kan senior dan junior, junior nya cenderung nurut sih. Ada diskusi juga. Satu lagi kan *backend*. Nah dia mau nyobain frontend juga. Saya invoke di Javascript-nya. Jadi saya ada *pairing* juga sama yang *backend*. Pair Programming memang kayak gitu. Pair Programming kan harus 1 monitor ya. Kenapa pakai ini itu memang terjadi. Ya diskusi saja sih. Ga sampe konflik. Coding: Tidak ada konflik dalam Pair Programming, Pair Programming dilakukan oleh seorang senior dan seorang junior.
H1.28	T: Berarti ga ada dampak dari Pair Programming ini? N: Iya kan tujuannya untuk *improve* yang terbaik. Dan itu memang Sprint itu.

	Coding: Pair Programming bertujuan untuk meningkatkan kualitas *code*.
H1.29	T: Kalau frekuensinya, sering ga Pair Programming. N: Enggak sih. Kalau dia memang butuh benar-benar bantuan, ya kita pair. Orang dari *backend* mau pindah *frontend*, mau *sorting* di *table*, pertama kita *pair* biar dia terbiasa, lalu kita lepas. Coding: Jarang dilakukan Pair Programming.
H1.30	T: Pengembang-nya ada *unit testing* ga? N: Ada. *Backend* dan *frontend* ada *unit testing*. Coding: Ada *unit testing*.
H1.31	T: Ada dokumen *testing* nya ga? N: Kita langsung ke *code* sih ga pakai dokumen. Ada *testcase*. Coding: Testcase sebagai dokumen *testing*.
H1.32	T: Kalau *stand upmeeting* sudah efektif belum sih? N: Kalau teori sih, waktu harus 15 menit, dan bicarain apa yang kita kerjain, kendala apa. Kalau sekarang sih memang ada beberapa yang *missed*. Misalnya mau ngomongin *feature*-nya, jadi malah panjang lebar disitu. Yang lainnya jadi bengong. Harusnya kan sebentar-sebentar saja kan. Masih sering terjadi. Coding: Daily Scrum masih kurang efektif.
H1.33	T: Biasanya berapa lama? N: Ga sampe ½ jam sih. Tapi kalau dia ngomongin teknis dan yang lain ga *involve* bosen saja. Pengen duduk lagi. Coding: Daily Scrum tidak efektif dalam segi pembahasan.
H1.34	T: Disini kan ada tim yang masing-masing pakai Scrum. Proses Scrum nya sama ga setiap tim? N: Saya belum pernah pindah tim kan jadi saya ga tahu. Tapi kalau saya lihat *board* nya sih beda. Kalau saya kan ada *to-do*, *progress*, *testing*, *done*. Ada tim lain yang *testing* by PM, *testing* by *designe*r. Kalau *daily standup* nya sama. Ada *board*-nya juga.

	Coding: Proses Scrum antar tim berbeda, Scrum *board* antar tim berbeda.
H1.35	T: Definition of done? N: Saat QA-nya sudah *testing* dan ga ada *bug* menurut *testcase* dia. Kadang memang QA *missedtestcase*. Jadi di *production* ada *bugs*. Jadi kita harus benarkan Coding: *Definitionofdone* jelas.
H1.36	T: Kan ada Sprint Planning. Dibobotin kan. Pembobotannya pakai apa? N: Angka, 1 3 5 8 13. Coding: Pembobotan backlog menggunakan skala angka (Planning poker).
H1.37	T: Ada *velocity*-nya ga sih? N: Pas pertama-tama kan pecobaan. Pas pertama itu bobotnya gede. Jadi beberapa *story* ga selesai minggu itu. Nah dievaluasi. Kapasitas kita segini. Kalau Sprint kedua masih terjadi ya berarti kapasitasnya harus dikurangi. Coding: Ada *velocity*, Sprint Retrospective digunakan untuk mengevaluasi *velocity*.
H1.38	T: Kakak nyaman ga sih kerja di anggota tim yang jumlah nya 5 ini? N: Nyaman sih. Tapi lebih baik jika ada *junior* ditemani sama *senior*. Kalau saya kan *frontend* sendiri di tim, jadi diskusi nya sama tim lain. Kalau *developer* kan ada *pullrequest*. Nah, saya *pullrequest* ke tim lain. Sekarang kan 1 *backend* nyentuh sebagian *frontend* juga, jadi saya bisa *pullrequest* ke dia jga. Coding: Anggota tim merasa nyaman bekerja di tim yang anggotanya tidak terlalu banyak.
H1.39	T: Kalau Business Analyst ada ga? N: Mungkin ada, tapi ga tahu itu sebutannya Business Analyst ga. Yang pasti di *happy* data ada 3 orang bisnis, dan 1 buat *design*er orang bisnis. Coding: Ada Business Analyst.
H1.40	T: Untuk masalah *skill* komunikasi, ada ga sih anggota yang *skill* komunikasinya kurang dan berdampak ke *development*-nya. N: Saya *introvert*. Kita kan *multi-culture* ya. PM-

	nya kan bule orang jerman. Agak malas sih ngomong langsung. Jadi saya lewat perantara *lead*. Dan misalnya orang bisniskan orang jerman juga, jadi kalau kesulitan komunikasi, ya minta bantuan. Ini *lead*-nya juga merangkap Scrum master. Coding: Ada anggota yang mengalami masalah komunikasi, kurangnya *skill* komunikasi.
H1.41	T: Setiap Sprint kan ada *value* yang dihasilkan. Ada dokumen untuk apa yang dikerjakan ga? N: Ga ada. Kalau buat *developer* ga ada. Ga sampe *databasedesign* atau *flowdiagram* gitu. Takutnya kurangi waktu kalau buat itu. Atau itu kerjaan siapa ga tahu. Coding: Tidak ada dokumentasi produk.
H1.42	T: Kalau ada orang baru, dijelasin *product*-nya gimana? Pakai dokumen atau manual? N: Kalau tim saya belum ada. Atau dikit banget. Misal pakai java, javascript API. Framework nya apa. Tapi ga detail banget. Tapi di tim lain ada yang pakai dokumen. Coding: Ada beberapa tim yang menggunakan dokumentasi.
H1.43	T: Lalu kalau koordinasi antar tim? Gimana koordinasi nya? N: Tim itu kerjain bagian masing masing. Tapi ada *meetinglead*. Ga tahu sih itu buat perusahaan atau gimana. Coding: Koordinasi tim dilakukan dengan melakukan *meetinglead*.
H1.44	T: Kalau QA Test nya gimana? N: Setiap *story* yang sudah masuk *testing*, seharusnya dia sudah bisa *test*. Coding: QA melakukan *testing* saat *story* masuk ke kolom *testing*.
H1.45	T: Disini ada PM Yng khusus nge-guide tim? N: ya, memang ada. Coding: Ada *dedicated* PM.

No	**Nama: Rizky Maulana** **Jabatan: Backend Engineer**

	Scrum Role: Development Team
H2.1	T: Apa risiko pakai Scrum? N: Pertama, kan cepat. Segalanya harus cepat. Jadi ada yang tidak terdeteksi saking cepatnya. Misalnya kita mau *developfeature* di Sprint. Tapi hambatannya tidak terdeteksi. Tapi untungnya Agile itu kan fleksibel ya. Kalau dari timnya ngobrol, ga jadi masalah. Tapi kalau tim yang pendiam, ga aware, Bisa kacau. Tapi selama timnya *responsive*, ga masalah. Tapi intinya dari kecepatannya itu jadi nya ga terdeteksi di awal. Kita cuma define ABC, ternyata ada DEF. Coding: Proses dalam Scrum berlangsung dengan cepat, sering muncul hambatan yang tidak terdeteksi di-*planning*.
H2.2	S: Berarti kalau ada *story* yang ga selesai, diterusin ke *next*Sprint? N: Ada kemungkinan *next*Sprint. Atau *story*-nya bobotnya gede, contoh paling gampang itu edit form *user*. Mau masukin *addressmap*. Kalau ga selesai, ya mungkin *edit*-nya jadi, tapi *maps*-nya di-Sprint minggu depan. Tapi *releaseedit*-nya, jadi dipecah. Coding: Story yang tidak selesai dilanjutkan ke Sprint selanjutnya.
H2.3	T: Banyak ga *meeting* selain *meeting*Scrum? N: Selama terjadwal sih ga masalah. Selama sesuai jadwal oke. Kalau berubah-ubah yang mengganggu. Dan seharusnya *meeting* itu ga boleh lebih dari 1 jam. Coding: *Meeting* diluar Scrum tidak menjadi masalah apabila terjadwal dengan baik dan durasinya tidak lebih dari 1 jam.
H2.4	S: Pernah 1 jam? N: Pernah, karena banyak yang diomongin. Coding: Waktu rapat dapat menjadi lama karena banyaknya hal yang harus dibahas.
H2.5	T: Ini kan lumayan mengganggu. Penyebab banyaknya *meeting* yang ga terjadwal di HappyFresh. N: Ga ga terjadwal sih. Cuma perubahan jadwal saja. Coding: Sering terjadi perubahan jadwal *meeting*.
H2.6	T: Dampak nya jadwalnya berubah jadi ganggu konsentrasi ga?

	N: Enggak sih. Cuma ganggu *planning* saja. Jadi *planning*-nya hari ini jadi besok gitu. Coding: Perubahan jadwal *meeting* tidak mengganggu konsentrasi *developer*, perubahan jadwal *meeting* mengganggu jadwal *developer*.
H2.7	T: Kalau retro, di HappyFresh gimana? N: Awalnya enggak. Baru 4 bulan saya masuk baru 2 kali retro. Entah kenapa itu tanya Scrum master. Coding: Sprint Retrospective tidak berjalan dengan rutin.
H2.8	T: Ada dampak nya ga antara ga retro sama ada retro? N: Retro menurut saya bagus. Kayak introspeksi, dan apa yang bisa kita lakukan kedepannya. Mungkin dari *creationtest*-nya jadi di-*include* kan kedalam tim. Coding: Sprint Retrospective berdampak positif pada tim Scrum, Sprint retrosepktif menjadi ajang instropeksi.
H2.9	S: Pernah ga ada masalah pribadi antar anggota tim? N: Kalau di tim sekarang enggak. Soalnya sekarang aktif. Coding: Tidak ada masalah pribadi antar anggota tim.
H2.10	T: Kalau masalah pribadi di bawa ke pekerjaan? N: Saya baru 4 bulan sih. Jadi selama ini sih belum ada. Coding: Belum ada masalah pribadi yang dibawa ke pekerjaan.
H2.11	T: Tujuan proyek kurang jelas? N: Yang dimaksud per Sprint? Iya, karena memang mungkin di-*grooming* ada yang kurang. Jadi ketika ada manager mengajukan *feature* ini, yang kurang adalah base kenapa harus ada *feature* ini. *Why*-nya. Jadi kadang-kadang kita ngerasa apa sih tujuannya. Jadi kayak kita ngeremehin PM. Karena ga dikonfirmasi, jadi kayak berpikir ini kayaknya asal ya. Jadi kerjainnya malah setengah hati. Ah nanti saja paling di-*rollback*. Nanti juga paling di-*drop*. Coding: Tujuan proyek kurang jelas, PO kurang

	mendeskripsikan tujuan proyek kepada *developer*.
H2.12	S: Berarti harus *confirm* dulu ya PM nya? N: Betul. Harus ada *story* telling. Jadi gagasannya kuat. Coding: Diperlukan alasan yang kuat untuk tiap backlog yang dikerjakan.
H2.13	T: Sering terjadi? N: *So far* ga ada *storytelling*. Cuma ada ya kayak dari apa ya perkiraan atau *address*. Tujuannya untuk mempermudah. Selama ini sih saya *missed*. Ga tahu yang lain. Coding: Beberapa *developer* kurang bisa mendapatkan maksud dari proyek yang disampaikan PO.
H2.14	T: Kalau *requirement* ga jelas? N: Pasti. Itu kan Agile ga sebanyak dokumen *requirement* sih ya. Intinya *define* apa tujuan yang mau dikejar. *Requirement*-nya nyusul. Bahkan kadang suka nambahin kayaknya kurang disini nih. Tapi kan Agile ga *static* ya. Jadi gitu. Coding: Sering terjadi perubahan kebutuhan.
H2.15	T: Disini kan PM megang 1 tim ya. Ada kemungkinan *overlaprequirement*? N: *Requirementoverlap* enggak, tapi kalau codingan bentrok ada. Karena setahu saya di PM pun ada *grooming* sebelum masuk tim. Coding: Tidak ada *ovelap* kebutuhan, terjadi bentrok pada *code* yang dibuat.
H2.16	T: Kalau masalah *dependency* tadi, jadi nungguin ini selesai dulu baru yang lain. Ada? N: Story kemarin ada. Coding: Terjadi Dependency.
H2.17	T: Penyebabnya? N: Karena memang ga dianalisa diawal apa sih *impact* di-*code*-nya. Pokoknya *feature*-nya jalan saja. Ga ada yang bentrok. Tapi implementasi di lapangan ada yang bentrok. Agak sulit sih di *define* kalau dari PM, harusnya di *grooming*-nya ada perwakilan *developer*. Kayaknya ada sih, mungkin karena ga terdeteksi ya. Karena cuma ngomong saja ga detail.

	Coding: Dependency terjadi karena kurang matangnya analisis saat *planning*.
H2.18	T: Selama kerja disini sudah berapa kali? N: Baru sekali. Coding: Jarang terjadi *dependency*.
H2.19	T: Ada Technical Debt ga? N: Ga ada sih. Coding: Tidak ada Technical Debt.
H2.20	T: Pair Programming? N: Ada awal masuk. Eh Technical Debt nya secara tim atau individu? Coding: Pair Programming dilakukan saat awal anggota baru.
H2.21	T: Tim. N: Kalau yang individu, karena saya basic nya java, tapi disini pakainya rels, awal-awal nya ada Technical Debt buat adaptasi. Coding: Pair Programming digunakan bagi anggota baru untuk beradaptasi.
H2.22	T: Pernah ngerasa ga sih ga cocok pair. N: Karena saya baru sekali, ya cocok sih. Coding: Tidak ada masalah ketidakcocokan dalam Pair Programming.
H2.23	T: Ad *unit testing* ga? N: Ada. Coding: Pengembang melakukan *unit testing*.
H2.24	T: Ada dokumen *testing*nya atau define sendiri. N: Self sih. Kan dipisah *backend* dan *frontend*. Kadang *scenariobackend* di-*define* sendiri tapi berdasarkan *story*. Tapi, ya kira-kira kita mau *input*-an a, b, c atau a *minus*. Dibuat sendiri. Itu buat lebih *confident* saja sih. Coding: Pengembang membuat dokumen *testing* sendiri berdasarkan *story*.
H2.25	T: Kalau *stand upmeeting* disini jam berapa? N: Setiap tim beda. Kalau saya jam 11.

	Coding: Waktu Daily Scrum setiap tim berbeda.
H2.26	T: Sudah efektif? N: Sudah. Coding: Daily Scrum sudah efektif.
H2.27	T: Berapa lama? N: Ga lebih dari ½ Jam. Paling 15 menit. Kadang cepat juga kalau ga ad kerjaan. Coding: Waktu Daily Scrum sudah efektif.
H2.28	T: Proses Scrum antar tim nya sama ga? N: Kalau *step*-nya sih semua sama ya. Coding: Proses Scrum antar tim sama.
H2.29	T: Disini 1 Scrum master pegang 1 tim atau pegang semua tim? N: 1 pegang banyak. Baru 1 SM. 1 Lagi masih magang. Coding: Seorang Scrum master dapat meng-handle beberapa tim sekaligus.
H2.30	T: Ada *definition of done*? N: Ketika sudah *meetacceptancecriteria*dari PM. Done itu ada dari *developer*, atau boleh dirilis. Kalau sudah boleh di-*release* harus sudah di-*test*dulu. Kalau done *developer* berdasarkan *accpetancecriteria*. Baru masuk test QA. Coding: Sudah ada *definition of done* yang jelas.
H2.31	T: Yang di awal kan ada pembobotan untuk masing-masing *story*. Pembobotan ini pernah *overestimate* ga? N: Ada. Karena Sprint awal *complex*. Kita jadi *over*. Malah rendah bobotnya. Jadi sampe 2 Sprint. Coding: Terjadi over*estimate* terhadap *story*.
H2.32	T: Kalau masalah *skill* komunikasi anggota tim, ada ga yang kurang bisa komunikasi, akhirnya ganggu proses *development*? N: Kalau tim lain sih ga tahu. Tapi tim saya sih aktif. Coding: Tidak ada masalah komunikasi dalam Scrum.
H2.33	S: Berapa orang 1 tim kakak?

	N: Frontend 4, Backend 2, QA 2, 1 Product Manager. Coding: -
H2.34	T: Untuk *product* akhirnya ada dokumen khusus ga? N: Saya ga tahu kalau itu. Dokumen dalam bentuk apa yang sudah di *release* begitu? Itu *scopejob*-nya PM dan SM sih. Biasanya kalau sudah *release* app, diinformasikan di *email*. Coding: Dokumen produk akhir diurus oleh PO dan Scrum master.
H2.35	T: Kalau masalah, kolaborasi QA dan DEV? N: Yang bisa nentuin di *test* kan *developer* dulu. Kalau sudah di *done* ke *test* berati sudah bisa di *test.* Coding: Test QA dapat dilakukan jika *developer* sudah selesai mengerjakannya.
H2.36	T: Kalau disini PM nya memang megang 1 tim ya? N: Iya. Coding: Ada *dedicated* PM untuk tiap tim.

No	**Nama: Artanto Ishaam** **Jabatan: Project Manager** **Scrum Role: Scrum Master**
H3.1	T: Apa risiko Scrum dari pandangan SM? N: Risiko pasti ada. Terutama untuk mereka-mereka yang masih belum paham dari *mindset*Agile sendiri. Scrumnya kan *framework*, Agilenya kan metodo*login*ya sendiri, lebih ke *mindset*-nya. Jadi saat kita ngejalanin Scrum, Agile itu kan dari *agility* ya, fleksibel gampang berubah, bisa adaptasi dengan cepat, kalau mental-mental orang nya masih metal yang lama, jadi kalau gue kerjain perubahan, jadi kerjaan tambahan buat gue. Intinya orangnya cari aman lah. Mau stabil-stabil saja, aman-ama saja. Mau pakai Agile atau Waterfall, ya dia pasti akan *decline*. Jadi kalau dari risiko sih paling tinggi disitu. Kalau lo masih orang dengan *mindset* lama, ga mau belajar hal baru, *engineeringpractice* baru. Karena Scrum dan Agile mereka kan ga menyentuh *engineering* practice. Mereka ngomongin *people*, dan process. Padahal ngejalanin Scrum saja tanpa didukung oleh *engineeringpractice* juga bakalan *failed*.

	Coding: Praktisi Scrum belum memahami *mindset*Agile.
H3.2	S: Cara menanamkan *mindset*Agile gimana? N: Kalau cara menanamkan *mindset*, yang jelas, *as*Scrum master kan dia kayak penjaga gawang. Pertama kali masuk pasti *on boarding* dulu. Samain persepsi dulu. Jadi ada *mini training, one day di train*. Belajar *culture*-nya di HappyFresh gimana. Biasanya kita by learning seiring waktu saja. Jadi kalau ada *mindset* yang salah ya dipukul balik pas retro. Jadi as Scrum master kita ga mengarahkan, tapi kita kasih jalan. Kita ga maksa dia ikuti jalan kita. Tapi kita giring dia. Jadi disini sudah lumayan jauhlah perkembangannya. Coding: Melakukan *on boarding* untuk menyamakan persepsi orang baru terhadap *mindset*Agile di perusahaan, menggunakan *Retrospective* untuk memperbaiki kesalahan *mindset* anggota.
H3.3	T: Setiap perusahaan kan beda-beda implementasi Scrum nya. Kalau di HappyFresh gimana? N: Kalau dibilang, Scrum nya beda-beda, gini. Scrum kan *framework*. Kalau di ibaratkan sepakbola, Scrum itu strategi, strategi sepak bola gitulah. Jadi mereka mungkin melakukan peraturan yang sama, tapi mungkin ada yang gelandang nya 2 atau 3. Itu balik lagi ke keadaan tim nya. Berhubung PO nya baru beberapa, jadi kita punya banyak tim tapi PO nya disharing beberapa tim. Yang beda kalau di Scrum kan ada Sprint Review. Sprint Review itu yang own PO dan dia *invite* beberapa *stakeholder*. Saat ini belum jalan. Yang jalan, adalah kita *review* tapi *review* bersama PO. Dan ada beberapa hal yang beda dari Scrum kita. Ga semua nya kita laksanakan di *planning*. Kita ada backlog grooming. Ada masa *developer* duduk bareng dengan PO. Dulu kita *planning* 4 – 5 jam, sekarang kita pecah waktunya supaya jadi efektif. Dan yang paling penting, Scrum itu metodologi untuk lebih Agile. Kalau kita bertahan dalam 1 *framework* berarti kita ga Agile. Coding: Penerapan Scrum di tiap perusahaan berbeda, Scrum hanyalah sebuah *framework* yang fleksibel.
H3.4	S: Sekarang misalnya ada anggota baru yang belum pernah pakai Scrum. Cara kakak latihnya gimana? N: Pertama pasti akan diajari prosesnya gimana. Terus, bagaimana cara dia bisa menanggapi proses tersebut. Karena masing-masing orang beda-beda.

	Jadi kayak Daily Scrum, aduh gue males nih, ngapain sih. Terus Daily Scrum, nah, cara yang paling efektif, kalau tidak mendekatkan Daily Scrum itu ke ritualnya, kita naikin manfaatnya. Orang di luar mungkin tahu kalau Daily Scrum itu *update* pekerjaan. Kalau kita *updatebrokers*. Itu adalah saat dimana lo ngeluh kalau kerjaan lo susah. Atau mungkin lo ga bisa kerjain, dan harus di-*drop* sama PO. Kalau fungsi itu sudah dinaikin, otomatis *developer* bakalan ikut. Kalau gue susah, gimana gue bisa ngasih tahu kalau *feature* yang gue kerjain diluar estimasi dan butuh bantuan si A. Jadi harus diangkat manfaatnya dari ritual Scrum tersebut. Coding: Menambah nilai manfaat dari Scrum agar *developer* tertarik untuk berpartisipasi.
H3.5	T: PO nya itu Product Manager kan? N: Iya Coding: Product Owner disebut sebagai Product Manager.
H3.6	S: 1 Project kan ada beberapa Sprint. Dan 1 Sprint ada beberapa *userstory*. Pernah ga ad User Story yang ga selesai? Kalau ga selesai gimana? N: Kalau ga selesai di bawa ke Sprint selanjutnya. Coding: Story yang tidak selesai dilanjutkan ke Sprint selanjutnya.
H3.7	S: Diperpanjang ga? N: Kita ga prefer di perpanjang jadi gini, kita punya 1 *coreproduct* yang lagi dipegang 3 tim. Kalau misalkan 1 tim Sprint diperpanjang, maka itu akan ganggu Sprint tim yang lain. Karena kita main *release*. Jadi *release* nya kalau bisa di bundle jadi 1. Kalau saya pribadi lebih *prefer*, apa sih, kenapa sih harus diperpanjang waktu Sprintnya. Mending lempar saja ke *next*Sprint. Kecuali itu *feature* yang sangat urgent. Kalau gitu kita bakalan *put it as a top priority*, jadi yang bawah ketendang. Walaupun kadang masih salah estimasi. Kita masih *keep learning* sih di bagian itu. Coding: Perpanjangan waktu Sprint akan mengganggu kinerja Sprint tim lain.
H3.8	T: Tujuan proyek kurang jelas? N: Ga juga. Selama PM bisa punya *vision* yang jelas ga akan ada yang gitu.

	Coding: Tujuan proyek jelas selama PO memiliki visi yang jelas.
H3.9	T: Kalau di HappyFresh sendiri gimana? N: Kalau di Happyfresh sendiri karena sudah punya *goal* buat naikin *userretention*, ya semua *story* akan mengarah kesana. Jadi sebenarnya, kalau dibilang tujuannya kurang jelas ga relevan sih. Coding: Tujuan proyek sudah jelas.
H3.10	T: Tapi *goal* itu ada *why*-nya kenapa gitu? N: Pasti ada. Coding: Product Owner sudah menjelaskan poin Why dari suatu tujuan proyek.
H3.11	T: *Requirement* nya ada ga yang kurang jelas? N: Requirement kurang jelas itu pasti akan selalu ada. Tapi bedanya kalau dialami oleh tim yang *mindset*-nya bagus, biasanya mau *requirement* jelas atau kurang jelas, akan cepat jelas secepat mungkin. Dia akan nanya, dia akan *speak up* dan tektok nya akan cepat. Beda dengan yang gini, yang kurang jelas juga ada toleransi nya. Ada yang kurang jelas yang ga bisa di prediksi. Yang sering ada itu, kurang jelas karena ada *case* yang sangat spesifik yang bisa membuat kejadian itu terjadi, nah, itu yang susah di prediksi. Coding: Kebutuhan sering kurang jelas, perlu adanya inisiatif dari *developer* untuk memperjelas kebutuhan yang kurang jelas.
H3.12	T: Kalau case yang ga jelas itu terjadi dampaknya gimana? N: Dampaknya, bagi kita biasa saja. Paling nanti dibelakang akan diomongin dengan PM. Coding: Kebutuhan yang kurang jelas membuat *developer* harus menanyakan ulang kepada PO.
H3.13	T: Pernah ga terjadi konflik *requirement* antar PM? N: Ga tahu sih. Tergantung gimana kita memandang konflik kayak gimana. Kalau kita konflik ngerjain 1 screen yang sama pernah. Tapi kalau 1 mau warna biru, 1 mau warna merah, ga pernah. Itu mereka kan selalu define duluan. Cuma kita ga tahu kalau ini lagi bikin A ternyata menyenggol B, ini juga bikin C ternyata menyenggol B juga.

	Coding: Tidak terjadi konflik kebutuhan.
H3.14	T: Dampak buat tim? N: Biasanya tim bakalan habis-habisan buat merging nya. Tapi kita pernah kejadian sih, lalu kita belajar, kita minta PM nya ngobrolin dulu biar ga nabrak. Coding: Sulit untuk menyatukan kebutuhan yang konflik.
H3.15	T: Kan awal Sprint ada priotas bobot. Perna over *estimate* atau under ga? N: Pasti. Nama nya di Agile, kita akan berusaha secepat mungkin untuk itu. Coding: Pernah terjadi over*estimate* terhadap *story*.
H3.16	T: Sering di HappyFresh gitu? N: Sering sih enggak. Cuma pasti 1 Sprintnya ada lah satu *story* ada yang kayak gitu, *missed-estimation*. Coding: Jarang terjadi miss-estimation.
H3.17	T: Pernah ga ada perubahan *requirement* di tengah Sprint? N: Pernah Coding: Ada perubahan kebutuhan di tengah Sprint.
H3.18	T: Kenapa bisa terjadi kayak gitu? N: Perubahan *requirement* itu banyak banget faktor nya. Bisa jadi karena, pertama ada. Gini, berubah ini berubah total atau ditambah atau dikurangin? Coding: -
H3.19	T: Merubah yang sudah ada. Bukan additional. N: Kalau merubah sih sudah pasti ada. Dimana pun Agile*organization*, *story* sudah jalan ke Sprint. Normal nya kalau dia bilang bahwa dia organisasi yang Agile, dia menjalankan *development* secara Agile, dia harus *agree* kalau di tengah jalan *story*-nya bisa berubah. Kenapa? Pertama, bisa jadi mereka menemukan ada case yang bolong yang belum mereka tanganin. Banyak sih faktor nya, misalnya dari segi desain, screen lah, atau belum memperhitungkan *lowconnection*. Terus, dari segi *product* sendiri, biasanya mungkin sudah ga *fit tomarket*, jadi ga

	usah dijalanin lagi *story*-nya. Atau sudah di implementasi, ternyata kita lihat ada perusahaan lain implemnetasi cara yang sama dan gagal, ya sudah deh diulangin. Atau mungkin lagi jalan, Sprint nya, mereka melihat Google Analytics. Ternyata yang lari ke *screen* itu ga banyak, ya sudah lah ga usah di *explore*. Banyak lah faktor nya. Coding: Selalu ada perubahan di Agile*organization*.
H3.20	T: Dan dampaknya *developer* mungkin harus siap. N: Pengembang harus siap. *That's why* tadi saya bilang di awal, *mindset*Agile itu harus ada. Kalau masih berpikir pakai cara lama sih ga bisa Coding: Pengembang harus siap menghadapi perubahan kebutuhan.
H3.21	S: Kalau gitu menurut aku *requirement*nya masih kurang mateng ga sih kak. Jadi si PM itu benar-benar matengin dan lihat *market*-nya apa. Biar waktu *requirement*-nya lagi di proses ga salah. N: Itu sudah pasti akan dilakukan. Kita juga ga mungkin ada *backlog* tiba-tiba masuk ke *development*. Kita punya namanya *definition of ready*. Sebelum *story* itu bisa di-*develop*. Contohnya misalkan, backlog-nya sudah harus lengkap, *acceptance criteria* nya sudah harus ada, desainnya ada. Benar-benar dirembungin sekali dua kali. *That means* berarti itu sudah siap banget, ini bakal dijalanin. Masalah itu kurang mateng atau enggak, beberapa hal ada yang bisa kita prediksi di awal ada yang enggak bisa kita prediksi sebelum *developer* mulai *code*. Kalau kayak gitu, faktor yang itu kita ga bisa hilangin. Kita ga bisa hindari. Karena ada beberapa hal yang begitu muncul, setelah masuk di *code* secara *technically*. Kalau dibilang mateng, mau semateng apapun, Agile*organization* harus *ready* kalau ditengah-tengah ada yang harus diganti. Coding: Story memiliki kriteria sebelum bisa di-*develop*, *story* dapat berubah di Agile*organization*.
H3.22	T: Kalau masalah *standupmeeting*. Gimana *stand upmeeting* di HappyFresh. Apakah harus di inisialsasi oleh Scrum master atau gimana? N: Sekarang sih sudah mulai, kalau enggak ada Scrum master, sudah bisa *stand up* sendiri. Walaupun inisiasi masih ya, jadi tugas Scrum master.

	Coding: Development team dapat melakukan Daily Scrum tanpa Scrum master.
H3.23	T: Menurut kakak, sudah efektif daily *stand up* disini? N: Sudah efektif. Karena kelihatan dari kalau di kita, kita tuh paling haram kalau ada satu masalah nih kebawa sampe hari besok. Jadi disini, parameter daily *stand up* nya berhasil adalah brokers hari itu kelar hari itu. Paling yang jadi pikiran saya sih, waduh masih ada yang itu masih belum selesai. Harus selesai secepatnya. Coding: Daily Scrum sudah efektif.
H3.24	T: Disini kan kalau dari Scrum Master nya sendiri, kan ada Scrum*board* kan. Kan ada kolom *done*-nya. Ada definisi khusus ga untuk *done*-nya di HappyFresh? N: Ada. Coding: Ada *definition of done*.
H3.25	T: Masing-masing tim ada definisi masing-masing atau? N: Satu untuk semua Coding: Tiap tim memiliki *definition of done*yang sama.
H3.26	T: Definition done-nya apa? N: Done kita itu ready sudah siap *deploy* ya. Jadi *definition of done* kita adalah pertama sudah ada di *release* branch. Kedua, Sudah abis itu translation nya sudah masuk. Ketiga, dokumentasi *feature* sudah dibikin. *Testcase* sudah semua pas dari QA. Sudah lewat *review* dari PM. Seinget saya sih itu saja sih. Coding: Ada *definition of done* yang jelas.
H3.27	T: Kemudian, kalau masalah jumlah anggota Scrum nya, rata-rata setiap tim berapa ya? N: 7 – 8 Coding: Jumlah anggota tim sudah sesuai dengan ketentuan pada Scrum guides.
H3.28	T: Bagaimana hubungan jumlah anggota tim dengan proses *development* di Scrum sendiri. Apakah semakin efektif atau bagaimana? N: Tergantung *product* yang dikerjain dan tergantung

	dari tipe tipe pekerjaannya sih. Ada tim yang ngerjainnya mungkin Cuma *backend* doang. Ada yang ada *backend* ada frontend. Ya jelas makin banyak makin bagus. Supaya lebih bisa terbagi rata dari segi *story* dan orangnya. Kalau dari segi koordinasi sih saya bilang relatif karena banyak atau sedikit juga ya kadang sedikit juga lebih susah diatur, dan banyak juga kadang lebih gampang diatur karena ada temen lain yang bisa ingetin. Ya realtif sih, selama masih dalam 7 plus minus 2 ya anjurannya Scrum. Coding: Kefektifan jumlah tim dalam Scrum relative.
H3.29	T: Ada Business Analyst ga? N: Scrum ga punya Business Analyst. Karena tugas Business Analyst ada di PO. Coding: Business Analyst sudah merupakan tugas dari PO.
H3.30	T: Berarti ga perlu ada tambahan *role* khusus BA ya? N: Enggak. Kita Cuma pakai *role* yang ada di Scrum saja Coding: Role dalam Scrum sudah cukup untuk menjalankan *development*.
H3.31	T: Pernah ga ada komunikasi nya ga baik dan mempengaruhi proses *development*-nya. N: Kalau komunikasi sih kalau itu ga jalan, berarti tugas saya ga benar. Coding: Scrum master bertanggung jawab atas masalah komunikasi yang terjadi.
H3.32	T: Kalau masalah mungkin ada anggota yang communication *skill* ya kurang. Gimana? Jadi ga ngasih tahu kalau ada masalah. N: Ya harus dipaksa. Itu aturan disini, kalau lo ga nyaman ya harus ngomong. Kalau yang ga kuat biasanya cabut dengan sendirinya. Masalahnya itu caranya. Kalau ga mau komunikasi ya silahkan kerja dengan sistem Waterfall yang sudah ter-*define* semuanya. Tinggal kerjain saja sesuai yang ada. Coding: Scrum memaksa anggotanya untuk dapat berkomunikasi dengan baik.
H3.33	T: Kalau antar anggota tim Scrum gimana komunikasinya? N: Yang jelas untuk perwakilan kayak gitu enggak

	ada. Karena kayak bikin hierarkikal. Cuman yang kita sedang ada adalah sesi *sync up* antar *mobile* team. Jadi kita ada sesi *sync up* khusus IOS Dev dan Android Dev. Biasanya sih yang dibahas adalah *codeconvention*. Paling sering sih eh, gue senin depan mau buat *feature* ini nih, butuh API ini ini. Tim lu sudah ada belum. Kalau belum gue suruh tim gue buat. Atau gue mau ngerjain *screen* ini nih. Ini buat nge-*prevent* kita ngerjain hal yang sama. Coding: Melakukan sesi *sync up* antar tim untuk membahas *codeconvention*.
H3.34	T: Terakhir, masalah PM, Apakah setiap tim punya satu PM khusus untuk pegang 1 tim. N: Saat ini sih 1 tim 1 PM. Tapi ada beberapa tim yang belum punya PM jadi masih dipegang CTO Kita. Ada juga yang 1 PM pegang 2 Tim. Masih belum punya spesifik 1 Tim 1 PM sih. Coding: Terdapat 1 PM yang meng-*handle* beberapa tim.
H3.35	T: Menurut kakak, perlu ga masing-masing tim punya 1 PM khusus? N: Perlu. Coding: Perlu adanya PO khusus untuk tiap tim.
H3.36	T: Menurut kakak, PM nya dicampur dengan beberapa tim, ada dampak nya ga? N: Pertama sih kalau dia punya 1 PM khusus, ya jelas PM nya akan lebih fokus dengan 1 tim itu. Kedua, pengambilan decision akan lebih cepat. Kita kan ngomongin Agile ya, PM availability dan *developeravailability* itu sangat dipentingkan. Jadi kalau lagi mau ini, eh tahu nya PM nya lagi di tim yang lain. Itu lambat. Sebenar nya kita ga siap untuk Agile kalau gitu. Disini sih, masih bisa di paksakan. Cuma ya ga tahu kita nanti, sampai kapan bisa. Coding: PM yang meng-*handle* beberapa tim akan memperlambat kinerja tim.
H3.37	T: Dan disini, kenapa enggak nge-*hire* PM lebih saja. ? N: Sudah, kita memang lagi nge-*hire* PM. Cuma belum dapet yang bagus saja. Coding: -

No	**Nama: Dedi Setiadi** **Jabatan: Quality Assurance** **Scrum Role: Development Team**
H4.1	T: Sudah berapa lama kerja dengan *framework* Scrum? N: Baru 7 Bulan. Coding: -
H4.2	T: Ada ga risiko pakai Scrum? N: Pasti ada sih. Coding: Ada risiko yang harus dihadapi saat menggunakan Scrum.
H4.3	T: Kalau boleh tahu, apa saja risikonya? N: Risikonya berarti ya bukan keuntungannya. Mungkin dari kurang detail. Kurang detail dari segi *acceptance criteria* dll. Apa lagi ya. Berpotensi jadi banyak celah-celah banget. Coding: Kebutuhan kurang detail berpotensi menimbulkan *bugs*.
H4.4	T: Nah, kira-kira apa sih yang menyebabkan malah jadi kurang detail pakai Scrum? N: Kalau yang saya tangkap sih kurang detail itu dibalikan ke *developer*-nya lagi sih akan seperti apa mengerjakannya, bagaimananya, bisa di implementasi atau enggak, berapa lamanya. Itukan dibalikan lagi misalnya dari PM ke *developer*. Coding: Pengembang harus menentukan sendiri detail yang harus dikerjakan.
H4.5	T: Kalau celah *bugs*, kok malah bisa banyak celah *bugs* dengan Scrum? N: Karena dari awal kurang detail, jadi sampe ke QA itu kebanyakan harus nanya ke *developer* juga, ke PM juga. Terlalu banyak tek tok juga sih. Coding: Kurang detail menyebabkan QA harus bertanya ke *developer* dan PO.
H4.6	S: Berarti kalau kurang detail gitu, sebagai QA, kakak nanya lagi ke *developer*-nya? N: Iya, nanya ke PM juga. Jadi lebih banyak tektok dan diskusi juga. Coding: Menanyakan kembali kebutuhan yang kurang

	jelas kepada PO dan *developer*.
H4.7	T: Kalau *meeting*, kakak banyak ga sih mengalami *meeting-meeting* diluar Scrum? Selain Daily Scrum dan *planning*? N: Kalau itu enggak ada sih. Paling tektok saja, paling informal saja, tanya-tanya. Itu suka lebih dari 5 menit. Melebar sendiri si kadang-kadang. Coding: Ada *meeting* informal diluar Scrum.
H4.8	T: Kemudian, ada retro kan. Ikut retro juga kan? Menurut kakak, retro nya rutin ga? Berjalan dengan baik ga? N: Selalu rutin sih. Setiap akhir Sprint biasanya ada retro. Coding: Sprint Retrospective selalu rutin dijalankan.
H4.9	T: Dan menurut kakak bermanfaat ga sih retro nya? N: Bermanfaat banget sih. Kan tujuannya mengevaluasi kan. Biasanya memang ya berguna banget sih. Coding: Sprint Retrospective bermanfaat untuk mengevaluasi proses *development*.
H4.10	S: Scrum kan mengutamakan kerja tim. Selama 7 Bulan kerja disini pernah ga ada masalah pribadi sama tim Scrum nya sendiri? N: Kalau masalah ga ada sih. Belum pernah. Coding: Tidak ada masalah pribadi di dalam Scrum.
H4.11	T: Tujuan proyek kurang jelas. Kalau menurut sudut pandang kakak sebangai QA gimana? N: Bisa jadi iya sih. Coding: Tujuan proyek kurang jelas di Scrum.
H4.12	T: Iya ga jelas atau ia jelas. N: Kurang jelas. Coding: Tujuan proyek kurang jelas.
H4.13	T: Kenapa bisa kurang jelas? N: Mungkin sih PM nya ngasihnya terlalu umum. Untuk detail-detailnya belum ada. Ga ada *requirement* yang detail.

	Coding: PO memberikan tujuan yang terlalu umum.
H4.14	T: Berarti itu juga termasuk *requirement* nya kurang jelas? N: Iya mungkin terlalu umum sih. Bukan ga jelas. Coding: Kebutuhan terlalu umum.
H4.15	T: Dampaknya terlalu umum ini? N: Ya gitu tadi. Banyak celahnya. Kadang kelewat juga. Kadang sudah sampai *production* baru kelihatan. Beberapa kali saya dari kerja disini. Coding: *Bugs* baru diketahui ketika sudah sampai *production*.
H4.16	T: Sering terjadi? N: Beberapa kali sih. Coding: Sering terjadi *bugs* yang tidak teridentifikasi sebelumnya.
H4.17	S: Kalau belum jelas *requirement* nya kakak sebagai QA bagaimana? Nanya ke *developer* atau bagaimana? N: Biasanya nanya ke PM sih. Langsung. Keputusannya ke PM juga. Coding: Pengembang akan menanyakan kebutuhan yang kurang jelas kepada PO.
H4.18	T: Lalu, disini ada beberapa PM kan? Pernah ga terjadi konflik *requirement* antar PM? N: Kurang tahu deh kalau itu. Mungkin belum sih kalau dari QA nya mungkin belum ada efek. Coding: -
H4.19	T: Kalau diawal Sprint kan ada pembobotan *story*. QA ikut ga? N: Ikut juga sih. Cuma mungkin kita samain dengan *developer*. Coding: QA terlibat dalam proses pembobotan, QA menyerahkan pembobotan kepada *engineer*.
H4.20	T: Kalau perubahan *requirement*nya? N: Pernah sih. Sering terjadi. Bahkan sudah selesai dari *testing* pun masih balik lagi. Sering terjadi sih. Coding: Sering terjadi perubahan kebutuhan.

H4.21	T: Penyebabnya apa ya? Kok bisa berubah-ubah gitu? N: Mungkin ada efek tim lain atau bagaimana yang ngerjain *feature* lain. Biasanya ada efeknya. Lebih kesitu. Coding: -
H4.22	T: Dan ini sering ya. N: Bisa dibilang sering sih. Coding: Sering terjadi perubahan kebutuhan.
H4.23	S: Berarti kalau kayak gitu, balik lagi ke on proses ya? N: Iya. Balik ke *onprocess*. Atau ada tiket baru yang harus di kerjakan *developer*. Coding: Perubahan kebutuhan menyebabkan status *task* kembali ke *inprogress*.
H4.24	T: Kemudian kalau untuk *testing* ada dokumen khusus untuk *testing*? N: Dokumen paling kita *testcase* saja. Jadi ada *result*, *state*-nya kayak gimana. Dan *expectedresult*. Coding: Dokumen *testing* berupa *testcase*.
H4.25	T: Sebagai tim Scrum, dan sebagai QA di tim Scrum. Ikut ga *standupmeeting* nya? N: Selalu ikut sih. Coding: QA selalu ikut dalam kegiatan Daily Scrum.
H4.26	T: Sudah efektif belum *standupmeeting*nya? N: Sudah sih. Coding: Daily Scrum sudah efektif.
H4.27	T: Biasanya berapa lama? N: 10 - 15 Menit Coding: Daily Scrum dilakukan tidak lebih dari 15 menit.
H4.28	T: Lalu, untuk kakak sendiri memang baru di tim ini saja atau sudah pernah pindah tim? N: Baru tim ini saja sih. Coding: -
H4.29	T: Setahu kakak, proses Scrum di setiap tim sama

	atau beda? N: Sama sih. Sama saja. Coding: Proses Scrum setiap tim sama.
H4.30	T: Kan dev nih, *engineer*nya. Mereka baru bisa masukin ke*done* saat kapan sih? N: Dari *engineer* ya? Setelah *build* ya. Kan kalau disini apps ya, jadi kalau *build* nya sudah jadi sih, sudah bisa di-*download*. Coding: Sudah ada *definition of done* yang jelas.
H4.31	T: Menurut kakak sendiri. Lebih enak kerja di tim yang jumlah nya kecil atau jumlah nya gede? N: Kecil itu kira-kira berapa nih? Coding: -
H4.32	T: Atau selama ini kakak kerja nya di tim yang kayak gitu? N: Mungkin kecil disini 8 orang atau 7 orang gitu. Coding: Jumlah tim Scrum sudah sesuai dengan yang disarankan Scrum guide.
H4.33	T: Atau kakak sudah pernah kerja di tim yang gede sebelumnya? N: Belum pernah sih kalau di Scrum. Coding: -
H4.34	T: Kalau komunikasi antar anggota tim gimana? N: Lancar juga sih. Ga ada masalah. Coding: Komunikasi di tim Scrum lancar.
H4.35	T: Kalau masalah *skill* komunikasi orang kan beda-beda. Ada ga sih yang kayak gitu dan mengganggu proses *develop*ementnya? N: Mengganggu enggak sih. Coding: Kemampuan komunikasi tidak mempengaruhi proses *development*.
H4.36	T: Ada tapi? N: Enggak sih. Disini belum ada. Coding: Tidak ada masalah *skill* komunikasi.
H4.37	T: Lalu untuk *product* akhir nya ada dokumentasi

	khusus ga sih? N: Kayaknya sekarang lagi enggak ada. Coding: Belum ada dokumentasi untuk produk akhir.
H4.38	T: Dan menurut kakak sendiri perlu ga sih ada dokumentasi produk? N: Penting sih. Coding: Perlu adanya dokumentasi produk akhir.
H4.39	T: Penting kan. Kok ga ada disini? N: Dulu sempet ada sih. Cuma kan PM nya banyak yang baru. Jadi mungkin belum diterapkan lagi. Coding: Ada atau tidaknya dokumentasi tergantung dari kinerja PO.
H4.40	T: Dan dari sisi QA sendiri ada ga dampak dari ga ada dokumentasi ini? N: Sering sih. Paling pas kita kan kalau QA ada *regression test*. Dari test dari awal sampai akhir semua *feature* di-*est*. Kadang suka sering ada, apalagi *feature* yang lama ya, perbedaan *requirement* antara IOS dan android. Coding: Tidak adanya dokumentasi membuat QA kesulitan melakukan *regression test*.
H4.41	T: Ada ga sih yang khusus nge-*test* produk dari apps gitu? N: Ya betul. Coding: -
H4.42	T: Ada koordinasi ga sih dari tim Scrum? N: Pas *regression test* itu kan kita ngumpul. *Feature* apa yang baru. Coding: Koordinasi dilakukan saat *regression test* untuk QA.
H4.43	T: Dan kakak sendiri, sebagai QA ngetest nya pas kapan? S: Setiap satu Sprint atau 1 User Story? N: Eh, satu *story* sih. Kalau *developer*-nya sudah *done* biasanya bisa langsung di-*test*. Coding: test dilakukan setiap *story* selesai.

H4.44	T: Disini ada PM masing-masing yang megang 1 tim? N: Iya betul. Coding: Ada PO yang meng-handle setiap tim.

No	**Nama: Isnan Freauseda** **Jabatan: Tecnhical Lead** **Scrum Role: Development Team**
H5.1	T: Risiko Scrum? N: Beyond *estimation*-lah. Diluar dari yang sudah dibayangin sebelumnya oleh *developer*. Kan pasti akan molor. Yang tadinya kita *estimate* 2 minggu. Ternyata ga bisa dikejar 2 minggu. Itu risikonya. Cuma, gimana caranya kita *providesolution* dari risiko itu sebenarnya, jatuhnya ke masalah komunikasi juga kan. Gimana *stakeholder* itu menganggap Scrum itu sebagai apa. Jadi kita sudah *estimate* beberapa *story* dan beberapa lama, *stakeholder* menganggapnya sebagai apanya. Cuma itu sudah diluar dari Scrumnya mungkin ya. Atau masih masuk juga? Coding: Under*estimate* terhadap *story* yang dikerjakan.
H5.2	T & S: Masih N: Itu salah satunya. Terus, mungkin jadi begini. Karena tadi *stakeholder* itu menganggapnya sebuah estimasi itu adalah *fix*, jadi yang tadinya kita sudah pakai AgileScrum akhirnya berubah jadi kayak, ini dari sisi *develop*met-nya. Jadi kayak mini Waterfall yang harus kita kerjain setiap Sprint. Coding: Scrum disalah artikan sebagai mini Waterfall.
H5.3	T: Nah, dan ini terjadi di HappyFresh. Malah menjadi mini Waterfall? N: Kadang iya, kadang enggak. Kadang *stakeholder* kita juga kurang begitu paham dari sisi technicalnya. Dari sisi *development*-nya sendiri. Coding: Terkadang Scrum dipandang sebagai mini Waterfall.
H5.4	T: Dan dampaknya apa ya dari malah mengimplementasikan mini Waterfall setiap Sprint? N: Jadi kan, kalian tahu ini gak, bedanya Waterfall sama Agile itu?

	Coding: -
H5.5	T: Kalau Agile dia bisa berubah-ubah, fleksibel. Kalau Waterfall, kan dia *fix*. N: Kalau kita mau *compare* ini mini Waterfall atau Agile, itukan bedanya yang di-*estimate* berapa dan harus selesai semuanya berapa. Kalau Agile kan *once* ada *bloker*, atau ada *overestimation* atau *underestimation*, ini kan bisa adjust kan *either*Sprint nya dipanjangin atau ada yang di *dropstory* sebelumnya. Kalau pengalaman gua dari awal sih kita enggak *applygood*Scrum karena dulu timnya cuma ada 5 – 7 orang, *whiches* semua orang ngerjain hal yang berbeda. Jadi menurut gua ga *make sense apply*Scrum disitu. Karena *totally* beda. Coding: Ada kondisi yang menyebabkan Agile tidak dapat diimplementasikan, mini Waterfall dan Agile itu berbeda.
H5.6	T: Sekarang? N: *Along the way*, kita kan nambah orang, nambah *requirement*, nambah *feature*. Dari situ kita belajar kita sudah *applygood*Scrum atau belum. Kadang kita sadar, kalau kita belum apply dengan benar, ya kita coba benarin. Dari yang tadinya kita *estimate* 10 *story*, 30 *storypoint* terus kita jadi lebih fleksibel. Coding: Scrum dapat diimplementasikan dengan baik seiring dengan semakin lamanya organisasi sudah menerapkan Scrum.
H5.7	T: Kemudian, kalau kakak sebagai *technicallead* ya, ada ga sih *meeting-meeting* lain selain *meeting*nya Scrum? N: Ya ada. Coding: Ada *meeting* diluar Scrum.
H5.8	T: *Meeting* apa ya? N: Jadi kita ada beberapa make *thirdparty* atau *framework*. Kita juga ada beberapa yang kita perlu *sync up* dengan mereka. Gimana kita *adjust* data yang kita kirim. Coding: *Meetingthirdparty* sebagai *meeting* diluar Scrum.
H5.9	T: Menurut kakak, *meeting-meeting* seperti itu mengganggu *development* ga? Mengganggu fokus atau

	gimana? N: Tergantung sih. Kalau gua sendiri kadang iya kadang enggak. Mengganggu nya ketika kita lagi *incharge* di satu *story*, ya *of course* ada waktunya. Kita punya *timebox for each story*. Jadi pasti akan mengganggu. Cuma seharusnyakan semua komponen yang ada di dalam tim ga boleh ikut *meeting* yang seperti itu. Jadi misalkan ada *meeting* dengan amplitude salah satu *vendor* kita, kita ga perlu semuanya *join* disitu. Coding: *Meeting* diluar Scrum mengganggu *developer*, tidak semua anggota Scrum harus mengikuti *meeting* diluar Scrum.
H5.10	T: Mungkin perwakilan saja ya kak. N: Ya. *Meeting* yang lain sih, contohhnya *technicalleadmeeting*. Itu juga. Ya karena buat *lead* doang, ya jadi masih *fine*. Coding: Ada *meeting* khusus untuk para *teamlead*.
H5.11	T: Kalau retro, gimana sih retro di HappyFresh. Dan berjalan dengan lancar ga, dan rutin ga dilakukannya? N: *Sorry* gimana tadi? T: Kan di Scrum ada proses retro, retro nya rutin di jalankan ga? N: Rutin di *endof*Sprint. Coding: Sprint Retrospective rutin dijalankan.
H5.12	T: Menurut kakak, retro nya penting ga? N: Parameter-nya apa saja nih? T: Mungkin apakah dia bermanfaat. Atau gimana gitu. N: Enggak, menurut kalian sendiri nih, kalian belajar Scrum kan. Retrospective sendiri apa? T: Introspeksi apa yang kita telah buat. N: Jadikan kita ada retro pasti penting. Cuma penting ga penting nya kan punya level. Dari 0 – 10 lah *let say*. Kalau kita *as a team ga* perlu nge-*consider level* pentingnya 3 atau 5 itu penting. Yang jadi point nya adalah *developerbeing honest* sama *how* dia *feel* ngerjain Sprint itu gimana. Dan

	comfortable nya dia dengan *environment* dan semuanya lah yang ada di Scrum. Kalau misalkan di satu tim itu ga nge-*consider* hal hal itu jadinya ga penting. Jadi lu ngapain buang 2 jam yang lu butuh, lu *mad*, *sad*, *glad*. Lu sama sama tahu lu ga ngapa-ngapain. Ya itu. Cuma kalau menurut gua sih lebih kearah apa namanya *obstacle* supaya kita bisa bikin retropektif itu penting. Itu satu tadi. How honest si *developer* itu. Orang cuma nulis 1 atau 2 tapi ga tahu. Yang kedua dia harus bisa nge-*recognise* apa yang bisa bikin dia *happy*. Apa yang bisa bikin dia ga *happy*. Parameter itu kan juga harus di generalisir se-*young* munkin. *As long as* semua yang di tim itu tahu Scrum itu apa dan dia harus punya parameter, harusnya sudah tahu. Coding: Penting atau tidaknya Sprint Retrospective itu bergantung dengan *developer* itu sendiri.
H5.13	S: Kakak kan dari awal HappyFresh sudah disini nih. Sudah hampir 2 tahun kan. Pernah ga ada masalah pribadi antar tim Scrum gitu yang buat *slack*antar satu anggota dengan anggota lainnya N: Ada mungkin ada enggak. Tapi dari yang gua rasain sih enggak ada. Coding: Tidak ada masalah pribadi dalam Scrum.
H5.14	T: Nah, risiko nya yang pertama tujuan proyeknya sudah jelas. Kalau menurut kakak sendiri ScrumHappyFresh gimana? Sudah selalu jelas atau gimana? N: Kalau saya sendiri sih, kadang ada beberapa *project* yang kita ga bisa dapet *why* nya dari orang *product*. Jadi mungkin masalah komunikasi ya atau *whatever* lah ya. Coding: Masalah komunikasi membuat tujuan proyek menjadi kurang jelas.
H5.15	T: Dan itu berdampak? N: Berdampak ke produktivitas. Coding: Tujuan proyek yang kurang jelas berdampak ke produktivitas.
H5.16	T: Dan itu sering ga terjadi? N: Mungkin, skala 0 sampai 5 bisa 3 mungkin. Coding: Beberapa kali terjadi tujuan proyek kurang jelas.

H5.17	S: Menurut kakak, kalau tujuan proyek kurang jelas karena disini PM nya yang kurang jelas ngasih tahu waktu Sprint Planning atau gimana? Gimana sih ngatasin kalau PM nya kurang jelas itu. Tanya lagi kita mau ngerjain apa *requirement* nya apa atau gimana? N: Kalau itu mungkin jadi kalau disini ya. Kita sudah coba *apply* supaya tim itu sedekat mungkin jadi lebih gampang untuk komunikasi. Jadi dulu, kita ada *groomingplanning* terus selama *development* ya akan seterusnya ngikutin itu. Jadi ketika ada masalah yang seharusnya berubah atau salah *estimate*, dulu ya ga pernah bisa, ya intinya gini, kalau tadi *sorry* pertanyaannya apa? S: Kalau misalkan sebagai *developer*. T: Lebih ke tujuannya. S: Si PM bingung tujuannya gimana dari PM nya kalau sebagai *developer* gimana? N: Untuk nge-*avoid* itu atau gimana? S: Misalnya itu sudah pernah terjadi di HappyFresh. N: Kalau pernah ya pernah. Terus kita ya, kita akhirnya kan *set up* supaya gimana sih caranya supaya orang-orang dalam tim itu *communication*-nya *fluid*. Karena sudah *fluid*, untuk ngomong ke PM harusnya sudah gampang banget. Tinggal *self initiative* dari si *developer* akan harusnya saling kritis lah ya. Misalnya *developer* ga ngerti sama tujuannya harusnya dia sudah bisa nanya. Ya tadi itu. *Communication*-nya seharusnya sudah dibikin supaya enggak susah. Coding: Meningkatkan komunikasi antara PM dan *developer* agar tujuan proyek menjadi jelas.
H5.18	T: Kalau masalah *requirement*-nya ga jelas karena penggunnaan Scrum, sering ga sih terjadi karena penggunaan Scrum ini, *requirement*-nya malah jadi ga jelas. N: Hubungannya sama Scrum gimana ya maksudnya? T: Ini saya takut jadi malah jadi mengarahkan sih. Tapi ya mungkin apakah dengan menggunakan Scrum malah sering banget *requirement*-nya itu malah jadi ga jelas. Ga detail.

	N: Pertanyaannya lebih kearah apakah Scrum jadi penyebab segala kekurangjelasan dan kesalahpahaman nya? T: kurang lebih begitu. N: Mungkin seharusnya enggak ya. Tapi gua ga tahu sih. Karena gua ga pernah belajar teori Scrum. Cukup atau ga cuma itu? Coding: -
H5.19	T: Gapapa sih. Kan tergantung perspektif masing-masing. Lalu kalau disini kan banyak PM, pernah ga terjadi konflik *requirement*? N: Kalau penyebabnya pasti nya apa ga tahu. Seharusnya itu, orang dari *product* sudah menyiapkan *clear* line bahwa bulan ini kita punya target ABCD, kita akan punya *feature* DEFG, dan seharusnya dia sudah tahu ketika dia apply. Yang akan punya implikasi dengan *feature* b, harusnya *feature* a dikerjain dulu baru *feature* b. Ga bisa diparalelin kerjainnya, Keduanya saling ngefek satu sama lain. Jadi nya ya konflik. Coding: Terjadi konflik kebutuhan.
H5.20	T: Dan itu sering ga terjadi disini? N: Lumayan sering. Coding: Sering terjadi perubahan kebutuhan.
H5.21	T: Kemudian, kan biasanya di awal Sprint ada pembobotan setiap *story* dan diukur kan tim itu kira-kira mampu ngerjain seberapa banyak bobot nya setiap satu Sprint. Menurut kakak sering ga terjadi pembobotannya terlalu gede, over*estimate* atau malah *underestimate* yang bikin tim nya jadi nyantai di akhir Sprint? N: Under*estimate* itukan *story* yang seharusnya dikerjain di 4 Sprint, tapi kita kerjadi di 2 screen. Jadi ada 2 *screen* lagi yang kita ga pernah *aware*, Jadi ketika estimation, kita ga pernah tahu nih kalau ada 2 screen yang nge-*impact* juga ke *story*. Mungkin itu yang namanya *underestimation*. Yang dikerjain Cuma 1 atau 2 padahal ada yang lain. Kalau *overestimation*, malah jarang kejadian. Kayak misalkan yang seharusnya Cuma ada di 2 screen, tapi dia mikirnya ada di 10 *screen*. Ada di *client* ada di *server* juga.

	Coding: Jarang terjadi *overestimation* terhadap *story*.
H5.22	T: Dan yang masalah *underestimation* ini sering terjadi? N: Sering terjadi *compared with overestimation*. Coding: Sering terjadi *underestimation* terhadap *story*.
H5.23	T: Kalau masalah perubahan *requirement*, kan ini sebenarnya berkaitan dengan yang tadi. Kan ini sebenarnya berkaitan dengan yang tadi. Menurut kakak sendiri gimana? Apakah di HappyFresh sendiri *requirement*-nya sering ga berubah? Mungkin *story*-nya tiba-tiba berubah. N: Maksudnya pernah terjadi? T: Iya di HappyFresh nya. Perubahan *story* di tengah-tengah Sprint. N: Perubahan *story* jarang. Mungkin bisa di bilang pernah sekali dua kali. Cuma jarang. Coding: Jarang terjadi perubahan *story* di tengah Sprint.
H5.24	T: Dan kok bisa terjadi perubahan *story* di Sprint? N: Itu kan sebenarnya satu hal yang jarang banget terjadi. Kalaupun terjadi, ya memang ga tahu ya. Kayaknya sih, pernah kejadian sekali dua kali. Cuma kayaknya kenapa kejadian ga tahu. Coding: -
H5.25	T: Kalau Technical Debt nih, Ada ga sih proses Technical Debt disini. ? N: Kayak nya enggak ada. Coding: Tidak ada Technical Debt.
H5.26	T: Kalau Pair Programming? N: Kita itu dari dulu coba *apply*Pair Programming. Cuma *resource*-nya kurang banyak, dan *story*-nya selalu banyak. Jadi, akan susah lah, kurang resource saja sih sebenar nya. Kecuali yang memang di satu Sprint itu *story*-nya susah banget dan harus dikerjain. Coding: Kurang sumber daya manusia menyebabkan Pair

	Programming tidak dilakukan.
H5.27	T: Kalau kakak sendiri pernah melakukan Pair Programmingnya. N: Kayak nya sudah lama banget ga pernah. Coding: -
H5.28	T: Disini ada *unit testing* ga oleh *developer*nya. N: Di *backend* iya, di *client* enggak. Coding: *Backend* melakukan *unit testing*, *frontend* tidak melakukan *unit testing*.
H5.29	T: Lalu setiap hari kan ada daily*stand up* nih. Efektif ga *daily standup*nya? N: Efektifnya sebenarnya kalau misalkan kita punya, jadi misalkan 1 *story* itu punya counter part *backend* dan *client*. Sebenarnya itu bisa dilakukan tanpa *daily standup*. Dan naturalnya jangan di *daily standup*. Keuntungannya buat orang yang misalkan lagi mau *blocking*, dan yang satu nya lagi *free* mau bantu. Cuman, kadang kalau misalkan ini *story* nya panjang, dan orang orang sudah tahu yang dikerjain itu itu saja. Maksudnya problemnya ya orang lain itu ga perlu tahu masalah technicalnya apa sih. Dan kalaupun ada, itu seharusnya bukan *partdaily standup*. Pertanyaannya apa tadi? Efektif atau enggak bisa dibilang efektif dengan catatan. Coding: Daily Scrum menjadi ajang membahas masalah teknis.
H5.30	T: Kemudian, kalau proses Scrum nya sendiri di setiap tim sama atau enggak ya? N: Sama. Coding: Proses Scrum antar tim sama.
H5.31	T: Lalu kalau masalah *definition of done*, ada gak sih *definition of done* yang umum digunakan oleh HappyFresh? N: Dulu begini, kita *definition of done*nya, *developer* do *code*, test, *commit*, push itu done. Cuma along the way kan kita dari yang orang nya Cuma 5 terus nambah QA nambah kayak *user* nya makin banyak, desainnya juga berubah, kita pelan pelan nambah, maksudnya dari yang sebelumnya *definition of done* itu, *developer* done, lalu kita tambah juga pass QA, pass *design*, dan pass PM. Sebenarnya kan seharusnya ketika *story* sudah pass QA seharusnya

	kan sudah selesai dari term of *technicalfunctionality* nya. Cumakan ada yang secara desain ga cocok. Coding: Ada *definition of done* yang jelas.
H5.32	T: Bahkan PM juga harus di *pass*. N: PM Itu sebenarnya bukan *definition* itu. PM cuma confirm kalau PM test dan sesuai dengan *accpetancecriteria*, itu sudah oke. Coding: PM melakukan *test* sesuai *acceptance criteria*.
H5.33	T: Kalau jumlah anggota di tim kakak berapa? N: Jumlah nya 7. Coding: -
H5.34	T: Menurut kakak, lebih enak kerja di tim yang anggota nya kecil atau yang jumlahnya banyak. N: Menurut kalian lebih enak mana? Lebih banyak orang lebih gampang lebih banyak orang lebih cepat atau sedikit orang cepat tapi susah. Ya ga bisa dibilang ketika begini maka lebih enak atau enggak. T: Apakah relatif atau gimana? N: Relatif. Coding: Efektif atau tidaknya Scrum tidak bergantung pada jumlah anggota tim.
H5.35	T: Untuk komunikasi antar anggota tim. Menurut kakak sendiri komunikasi nya gimana kalau diHappyFresh. Di satu tim. N: Bagus atau enggak bagus maksudnya? Ya bagus kalau menurut ku. Coding: Komunikasi antar anggota tim Scrum bagus.
H5.36	T: Kalau seandainya ada orang yang kemampuan komunikasi nya kurang nih. Ada ga sih yang kayak gitu kurang kemampuan komunikasinya dan akhirnya berdampak ke *development*-nya sendiri. Jadi mungkin dia ada masalah tapi susah buat ngasih tahu kalau dia ada masalah dan ga dikasih tahu. N: Ada. Dan akhirnya sih ya ga dikasih tahu juga. Jadi kita tahu dia seperti itu ya kita biarin saja kayak gitu. Cuma kita juga kayak punya banyak *feedback* supaya dia *recover*. Ya kita masih tolerir

	lah. Coding: Ada orang yang memiliki kemampuan komunikasi yang kurang.
H5.37	T: Dan hal kayak gitu menurut kakak berdampak ga sih ke *development*-nya? N: Berdampak T: Dampaknya apa? N: Pasti *slow* down. Pasti ngelambat. Coding: Kurangnya kemampuan komunikasi anggota tim membuat proses *development* menjadi lambat.
H5.38	T: Lalu kalau di HappyFresh sendiri ada ga sih dokumentasi untuk produk akhirnya? N: Maksudnya dokumentasi setiap Sprint? Sekarang baru mulai ada. Coding: Sudah ada dokumentasi produk akhir.
H5.39	T: Perlu ga sih dokumentasi itu? N: Perlu. Dari sudut pandang siapa nih? T: Pengembang. N: Kalau *developer*, kayak nya sih malah ga perlu. Lebih perlunya malah kayak dokumentasi API, dokumentasi class. Coding: Pengembang kurang memerlukan dokumentasi produk.
H5.40	T: Lalu, untuk PO nya disini memang setiap tim itu ada satu PM khusus untuk menangani masing-masing tim? N: Ya Coding: Setiap tim memiliki satu PO.
H5.41	T: Dan untuk koordinasi antar tim nya apakah ada *meeting* khusus? N: Kita ada *platformsyncup*. Jadi IOS akan sync up dengan dev ios cross team dan dengan android juga. Coding: *Platformsyncup* sebagai upaya koordinasi antar tim *developer*.
H5.42	T: Dan QA nya selalu ngetest saat *task* nya sudah masuk ke *testing* QA?

	N: Iya. Coding: QA melakukan *testing* saat *task* sudah masuk ke kolom *testing*.

No	**Nama: Rheza Sastra** **Jabatan: IOS Pengembang** **Scrum *role*: Development Team**
H6.1	T: Ada ga risikonya pakai Scrum? N: Apa ya. Palingan ya *underestimate*, pertama kan kita analisis kayak kompleksitasnya fungsi nya. Kadang-kadang kalau *developer*-nya orang baru, biasanya risiko nya akan tinggi. Dia ga tahu kan kayak gimana sih susah nya buat yang itu. Kadang-kadang kebanyakan orang *developer* baru kayak *underestimate* gitu, ah gampang nih. Tapi itu bisa di bagiin kayak misalnya 1 tim ada yang *senior* nya. Kayak gitu sih. Risiko nya ya bakalan kayak kalau kebanyakan baru ya susah juga. Coding: Pengembang baru mengalami under*estimate* terhadap *story*.
H6.2	T: Berarti mungkin supaya newbie-nya ini ga *underestimate*, harus ada yang dampingin. N: ya. Coding: Pengembang baru harus didampingi oleh senior dalam melakukan *planning*.
H6.3	T: Kalau *meeting*, kan di Scrum itu ada beberapa *meeting*, ada Daily Scrum, *planning*, retro. Ada ga *meeting* lain selain *meeting*Scrum? N: Ada sih Coding: Ada *meeting* diluar Scrum.
H6.4	T: Banyak? N: Lumayan. Coding: Banyak *meeting* diluar Scrum.
H6.5	T: Mengganggu ga *meeting* diluar Scrum nya? N: Kalau kebanyakan sih mengganggu juga. Tapi sejauh ini sih oke-oke saja. Coding: Terlalu banyak *meeting* akan mengganggu kinerja *developer*.
H6.6	T: Dan disitu juga ada retro kan? Retro nya rutin

	dijalankan ga? N: Rutin sih. Coding: Sprint Retrospective rutin dijalankan.
H6.7	T: Menurut kakak, penting ga sih retro nya dijalankan? N: Penting sih. Buat evaluasi. Coding: Sprint Retrospective penting untuk melakukan evaluasi proses *development*.
H6.8	S: Terus, kakak kan disini sudah satu tahun. Terus kerja secara tim di 1 tim Scrum. Pernah ga ada masalah pribadi gitu antar anggota tim nya. Yang menggaggu ke pekerjaan waktu *development*. N: Kalau masalah tim sih enggak sih. Enggak ada. Coding: Tidak ada masalah pribadi antar anggota tim Scrum.
H6.9	T: Apakah tujuan proyek nya ga jelas atau enggak pakai Scrum? N: Kalau menurut gua sih dari PM nya sih. Kalau dari *developer*nya sih jelas lah kita. Coding: Tujuan proyek jelas.
H6.10	T: Kalau *requirement*-nya gimana? Jelas ga? N: Kadang-kadang ga jelas. Tapi itu kebanyakan dari PM nya sih. Coding: Kebutuhan dapat menjadi tidak jelas.
H6.11	T: Kenapa PM nya? N: Ga tahu. Kadang-kadang *definestory*-nya banyak yang ga jelas. Itu malahan dari PM nya sih. Kalau dari *developer*-nya sih jelas jelas saja sih. Pengembangnya tahu dia mau buat apa saja. Ya *miss* komunikasi saja. Coding: PO membuat *story* yang kurang jelas.
H6.12	T: Itu yang nentuin *story* itu memang PM saja atau *developer* juga terlibat? N: PM sama *developer* kadang kadang bantu bantu. Coding: PO menentukan *story* bersama dengan *developer*.
H6.13	T: Kalau ada *story* yang ga jelas gitu dampaknya apa

	ya? N: Ke-*delay*. Jadi *delay*. Di awal kan *underestimate* yang newbienya gara-gara PM nya ga jelas *story*-nya. Kadang-kadang mereka mintanya ga jelas gitu. Misalnya gua mau yang kayak gini nih, tapi loading-nya kayak gimana. Kalau di *mobiledeveloper* kan ada *onlineoffline*. Kalau *online* kan *bestscenario*. Kalau *offlineworstscenario*. Itu yang kadang ga dipikirin sama PM nya. PM-nya less *experience*. Nah itu kenapa ya. Pertama ya PM-nya itu kurang pengalaman, inti nya itu saja. Off line kan harus dipikirin lah di *productdevelopment* gitu. Ada beberapa kali miss. Bayangkan misalnya lihat *analytic* buat naikin rating KPI mereka. Coding: Story yang kurang jelas membuat proses *development* menjadi lama.
H6.14	T: Disini pakai KPI juga? N: KPI nya mereka yang PM kan ada KPI nya tuh. Kayak *compression*-nya gitu. Coding: -
H6.15	T: Kemudian, disini ada beberapa PM kan? Pernah ga terjadi konflik *requirement* antar PM? Jadi entah *requirement*-nya overlap atau malah *dependency*. N: Kalau itu bukan tugas kami kali ya. Dari PM. Tapi sih pernah kayak gitu. Itu dari PM nya sendiri sih. Scrum itu kan metodologi Agile, kalau ga jalan di-*requirement* kan tanggung jawab PM kan. Coding: Pernah terjadi konflik kebutuhan karena PO.
H6.16	T: Untuk kan biasanya di Sprint Planning ada pembobotan gitu kan untuk setiap *story*. Itu yang ngebobotin *developer* ya? N: Pengembang. Coding: Pengembang melakukan pembobotan *story*.
H6.17	T: Pernah ga pembobotan nya *overestimate* atau *underestimate*? N: Kalau *underestimate* sih ga apa-apa ya. Kalau *overestimate* yang bahaya. Coding: *Overestimatestory* berbahaya terhadap *development*.
H6.18	T: Tapi pernah ga di HappyFresh?

	N: Pernah. *Underestimate* juga pernah. Kalau kita *overestimate* ya kita ambil *story* lain dari *backlog*. Coding: Pernah terjadi *overestimatestory* dan *underestimatestory*.
H6.19	T: Kalau *overestimate* itu penyebabnya apa? N: Ya, penyebabnya bisa banyak sih. Kita ga ada pas lagi jalan ada *bug* yang ganggu gitu. Jadi butuh *solve* dua hari satu hari. Kan kita cuma 10 hari kerja. Apa lagi ditambah *meeting-meeting* kan. Satu hari dua kali. Ada yang sakit juga. Jadi kan ga bisa di prediksi. Jadi kita sebisa mungkin handle yang *overestimate*. Coding: *Overestimate* terjadi karena *developer* perlu memikirikan hal hal tak terduga yang mungkin terjadi.
H6.20	T: Lalu, kalau *requirement* nya di tengah jalan berubah pernah ga sih gitu? N: Pernah saja. Coding: Terjadi perubahan kebutuhan di tengah Sprint.
H6.21	T: Kenapa bisa berubah di tengah jalan. N: Berubah disini dalam artian masih dalam satu *region* yang sama. Jadi kayak ada tambahan apa gitu. Kalau di HappyFresh ya. Coding: Perubahan kebutuhan berupa tambahan fungsi yang harus dikerjakan.
H6.22	T: Dan itu memang sering terjadi kayak gitu? N: Iya. Kalau sering iya di kita. Coding: Sering terjadi perubahan kebutuhan.
H6.23	S: Kalau sebagai *developer* gitu kalau ada perubahan *requirement* gitu ganggu ga sih kak? N: Ya ganggu lah. Coding: Perubahan kebutuhan mengganggu proses *development*.
H6.24	S: Kakak gimana kali sebagai *developer* menyikapinya? N: Ya kerjain saja. Coding: Perubahan kebutuhan harus tetap dikerjakan

	oleh *developer*.
H6.25	T: Disini ada Technical Debt ga ya? N: Ada kayaknya. Coding: Ada Technical Debt.
H6.26	T: Itu di-include dalam Sprint atau diluar? N: Kadang kadang didalam Sprint. Coding: *Technical Debt* dilakukan didalam Sprint.
H6.27	T: Ada masalah ga selama Technical Debt? N: Tergantung sih kalau ada *bug* yang penting ya kerjain. Bahkan *bug* yang gede masuk Sprint juga. Coding: *Technical Debt* dilakukan untuk memperbaiki *bug*.
H6.28	T: Lalu, kalau Pair Programming pernah? N: Pernah. Coding: Pernah melakukan Pair Programming.
H6.29	T: Pernah ga sih ngerasa ga cocok *pairing* sama dia? N: Cocok-cocok saja sih. Coding: Tidak ada masalah ketidakcocokan saat melakukan Pair Programming.
H6.30	T: Lalu, sebagai *developer* ada ga sih proses *unit testing*? N: Kalau dari kita ga jalan. Kalau dari ios *unit testing*-nya ga jalan. Cuma UI *testing* nya saja. Kalau yang di *backend* nya *unit testing*-nya jalan. Kalau di frontend nya di ios dan android, *unit testing*-nya ga jalan tapi ada UI *testing*. Coding: tim *backend* melakukan *unit testing*, tim frontend melakukan UI *testing*.
H6.31	T: Yang test UI nya *developer* sendiri? N: Kalau saya sih saya buat sendiri. Coding: Pengembang melakukan test sendiri terhadap UI produk.
H6.32	T: Itu pakai *testcase* juga? N: Pakai *testcase*. Coding: Pengembang membuat *testcase* untuk melakukan

	unit testing dan UI *testing*.
H6.33	T: Kalau *daily standup* nya, sudah efektif belum? N: Efektif saja. Cuma paling kesiangan. Murni *stand up* jam 11. Coding: Daily Scrum sudah efektif, waktu Daily Scrum terlalu siang.
H6.34	T: Itu memang komitmen nya. N: Ya sih. Ya iya. Coding: -
H6.35	T: Biasanya *stand up* berapa lama? N: 20 Menit paling lama. Tergantung apa yang dibahas. Coding: Waktu Daily Scrum bergantung pada apa yang dibahas.
H6.36	T: Pernah ga sih di *daily standup* malah ngebahas yang ga sesuai konteks. N: Maksudnya? T: Kan di *daily standup* itu ngebahas apa yang dilajukan, hambatannya apa, dan apa yang mau dilakukan. Pernah yang diluar itu? N: Enggak sih. Masih fokus. Coding: Daily Scrum sudah membahas hal yang sesuai konteks.
H6.37	T: Untuk *definition of done*, ada *definition of done* khusus ga? N: Ada sih. Ada 2. *Definition of done* kalau *developer* selesai kerjain. Kalau done itu benar-benar dari PM nya dan QA nya sudah oke. Coding: Ada *definition of done* yang jelas.
H6.38	T: Kalau masalah tim nya, semakin banyak tim menurut kakak gimana? N: Gimana maksudnya? T: Maksudnya semakin banyak anggota tim. Gimana? N: Enggak juga sih. Mending yang sedikit. Coding: Pengembang lebih suka bekerja di tim yang

	sedikit.
H6.39	T: Dan di HappyFresh, atau tim kakak dikit juga? N: Dikit. Pengembangnya Cuma 4. Kalau di tim lain sampe 6 atau 8. Kalau QA tambah lagi, QA ada 2 satu tim. Coding: -
H6.40	T: Terus antar anggota tim gimana komunikasi nya? N: Lancar. Coding: Komunikasi antar anggota tim lancar.
H6.41	T: Ada ga sih di tim kakak, yang mungkin kurang bisa komunikasi nih malah ganggu proses *development*? N: Ga ada sih kayak nya. Coding: Tidak ada anggota yang memiliki kemampuan komunikasi yang kurang.
H6.42	T: Untuk *product* akhir nya sendiri ada dokumentasi khusus ga sih? N: Ada tapi yang bikin bukan saya. PP (Scrum Master) itu yang bikin. Coding: Ada dokumentasi produk akhir, dokumentasi produk akhir dibuat oleh Scrum master.
H6.43	T: Kalau antar tim ada koordinasi ga? N: Ga ada. Ngomong saja langsung. Coding: Tidak ada koordinasi khusus antar tim.
H6.44	T: Lalu, QA itu memang selalu ngetest kalau dia sudah masuk ke *testing* nya QA. N: Iya. Coding: QA melakukan *testing* ketika *task* sudah masuk kolom *testing*.
H6.45	T: Dan setiap PM memang dikhusus kan megang 1 tim. N: Enggak. Bisa banyak. Tapi kalau Scrum sih 1 PM 1 Tim. Kayak nya ada deh 1 PM 2 Tim. Coding: Satu tim di-handle oleh satu PO.

No	**Nama: Hanifa Azhar** **Jabatan: Product Manager** **Scrum Role: Product Owner**

H7.1	T: Disini memang sebutan PO itu PM ya? N: Iya. Coding: PO disebut sebagai PM.
H7.2	T: Soalnya tadi agak bingung. N: Soalnya kalau di Scrum PO itu bisa siapa saja kan. Bisa orang dari bisnis. T: Iya N: Kalau kita itu PM tu nengahin itu semua. Jadi kayak bisnis *market*ing, *operation*, negara2 lain. Kan negara kita ada lima, Filipina, Thailand, Taiwan, Vietnam. Itu semua kalau ada permintaan, mereka ke kita dulu. Jadi kita yang prioritasin yang mana perlu dibuat gitu. Coding: -
H7.3	T: Terus kakak, kalau boleh tahu sudah kerja disini berapa lama? N: 4 Bulan. Coding: -
H7.4	S: Jurusan apa kakak? N: Teknik informatika. Coding: -
H7.5	S: Di? N: Di ITB. Tapi bukan yang jago ngoding. Jadi gini. T: Sama. Coding: -
H7.6	T: Kan kita ini mau identifikasi risiko nih. Oleh karena itu, selama 4 bulan kerja di *framework*Scrum, kita mau tahu apa risiko pakai Scrum? Dari sudut pandang dari Product owner? N: Risiko nya, wah berat. Sebenarnya risiko nya bisa dari 2 sisi. Karena kalau misalnya kita kan nyiapin, kalau Scrum dibandingin sama Waterfall ya. Biasanya kan Waterfall kita nyiapin sudah gede mau buat apa terus jadi. Kalau Scrum kan kecil. Kalau dari PM masalah nya adalah kalau kita belum nyiapian buat Sprint berikutnya. Karena disini prosesnya lumayan lama untuk buat *story*. Kita harus

	kumpulin data dari google analytic. Terus abis itu, dari beberapa *tools* lain kayak fc. Kita bikin misal kita temuin halaman ini jelek, jadi harus diperbaiki. Jadi kayak halaman A harus kita benarin karena *drop off*-nya gede banget. Orang keluar app dari situ. Terus kita buat *mock up* sama desainer, mungkin benarinnya kayak gini. Habis itu kita harus *user testing*. Dan itu proses nya 1 sampe 2 minggu. Setelah *user testing*, bisa jadi *user* nya bilang oh, dia ga ngerti ternyata *mock up* baru kita kan, Jadi kita harus perbaiki lagi. Itu proses lumayan lama, jadi ya risikonya adalah disaat kita belum nyiapin buat Sprint berikutnya. Sementara waktu buat nyiapin buat Sprint berikutnya bisa 2 minggu. Itu bahaya kalau tim nya sampe ga tahu mau ngerjain apa. Terus jadi dapet kerjaan yang sepele. Kan pasti moral tim nya turun kan. Kayak kenapa sih lu ngasih gua kerjaan kayak gini. Itu salah satu risiko. Tapi itu sebenar nya bisa di mitigasi sih. Karena kayak, sekarang sih kita nyiapinnya minimal 2 sampe 3 Sprint kedepan dari setiap *story* nya. Jadi mikirnya kayak jauh banget. Sampe ditanyain Sprint apa ngerjain apa, sudah lupa. Terus, risiko lainnya, kayak justru pakai Scrum komunikasi bagus banget sih. Ada *daily standup*. Nah risiko berikutnya kalau PM lagi banyak *meeting*, misalnya lagi ga bisa ikut *daily standup*, terus komunikasinya biasanya disitu. Kalau enggak kayak oh gua lupa ngasih, sudah ngobrol nih ada masalah tapi ga dikasih tahu ke gue karena gue lagi *meeting*. Terus gue baru tahu besoknya lagi, itu bisa jadi masalah. Sekarang sih nyobanya, baru beberapa minggu ini sih, gua lagi susah ikut *standup*. Lagi nyoba kayak lewat Slack gitu sih. Tapi untung nya sih, kita bikinnya kita duduknya deketan banget. Jadi kalau ada masalah langsung ngobrol. Kalau dari proses Scrum nya sendiri apa ya risikonya. Paling kalau Scrum yang gue baca sih kan kayak harusnya Sprint sudah jalan ga bisa ngubah kan. Kayak lu ga bisa masukin *story* baru. Karena kaya ganggu kan. Kalau disini, si PP (Scrum master) percaya banget sama Agile gimana lo harus banget. Yang penting kita adaptasi sama perubahan kan. Jadi kalau misalnya ada yang jauh lebih penting, ya kita sebagai PM harus bikin tim percaya ini lebih penting. Terus jelasin kenapa baru bisa masuk. Cuman ga enak saja. Coding: Waktu PO untuk membuat *story* lama, Waktu Scrum yang terbatas membuat Product Owner kesulitan

	untuk menentukan PBI untuk Sprint selanjutnya, Turunnya moral *developer* apabila mengejakan pekerjaan yang kurang bernilai, ketidakhadiran Product Owner pada Daily Scrum akan membuat *miscommunication*.
H7.7	T: Kemudian, untuk ini kan sudah bisa dimitigasi semua kan. Kalau yang harus siapin *next*Sprint itu sudah di pikirkan dulu jauh-jauh hari segala macem. Kalau misalnya banyak *meeting* ini sudah pakai Slack juga kan? N: Tapi kurang ideal sih. Jadi sejauh ini paling bagus sih tim nya memang komunikatif banget jadi kalau mereka inget mereka langsung nanya. Pas gue sampai kursi. Tapi kalau enggak, iya baru inget besok nya lagi. Coding: Tim secara aktif berkomunikasi dengan PO.
H7.8	T: Kalau yang nentuin *Product Backlog Item* tugas PM? N: Iya. Coding: Product Owner bertugas menentukan Product Backlog.
H7.9	S: Gimana nentuin prioritas PBInya? N: Itu, paling seru sih kerjaan nya. Jadi kayak misalnya kita mau bikin misalnya ada 2 *feature* baru. Satu replacement, satunya mau benarin *delivery*. Itu kan bisa banyak banget *story* kan. 1 replacement bisa ada 5 *story*, *delivery* juga 5 *story*. Dalam 1 Sprint lu ga bisa kerjain semuanya. Biasanya kita nentuin yang paling *impact* dulu. Jadi kayak yang *replacement* ini ternyata bisa ngurangin *dropoff*. Atau bisa ngasih *conversion*-nya lebih baik. Itu pasti kita kerjain itu dulu. Tapi kalau replacement ini bisa lebih bagus dari *delivery*, tapi *story* nya ada 15, jadi butuh 3 Sprint. Paling kita kerjain yang *delivery* dulu. Yang lebih cepat. Dan yang *replacement*-nya sambil kita benarin. Jangan 3 Sprint itu semua. Supaya lebih dikit. Tapi juga kadang-kadang ada *bugs*. Kalau *bugs* kita punya 1 *board* sendiri sama permintaan-permintaan negara yang suka aneh-aneh gitu. Kayak tambahin bendera dong disini. Kayak ga penting banget. Mereka kan bukan orang *product*, jadi kayak oh ini gampang magic gitu ya. Jadi kita punya 1 *board* sendiri untuk ngurusin itu. Jadi nanti itu semua masalah di situ di taruh di *board* itu dan di prioritas sama PM itu. Kalau si PM itu ngerasa ini penting banget,

	kayak *bugs* ini, kita nentuin *bugs* penting itu yang ngeblog *user* bisa belanja. Jadi kalau *bugs* ini bikin orang ga bisa belanja penting banget dikerjain. Kalau enggak, ya masih bisa di pending. Kalau dia nemuin gitu, dia kasih tahu PP. PP Bantuin tim mana yang bisa masuk kerjain itu. Coding: Prioritas PBI ditentukan berdasarkan *impact* yang diberikan tiap *story*. Memisahkan *board* untuk *bugs* dan permintaan negara dengan *board* untuk *productdevelopment*.
H7.10	T: Tujuan proyek kurang jelas. Pernah gitu ga? N: Iya sih. Contoh nya sih mirip yang kayak kalau kita belum nyiapin *story* untuk Sprint depan, mereka kayak ngerjain *story* yang ga jelas gitu. Pasti kita punya backlog yang isinya, mungkin tombol ini kurang warna ijo. Kita pasti punya backlog sepele itu. Tapi biasanya pasti ga akan pernah dikerjain. Karena kita punya *story* penting. Kalau kita ga punya *story* penting dan ngerjain itu, pasti mereka sedih. Tapi belum pernah separah itu. Coding: Tujuan proyek kurang jelas.
H7.11	T: Dan sering ga terjadi? N: Kalau gue pas awal masuk pernah sempet terjadi 1 Sprint. Gara-gara, itu memang belum siap sih. Jadi work yang gua kerjain itu gede banget ternyata dan abstrrak banget. Kita *user testing* 2 kali sampe belum dapet. Akhirnya kita ngerjain *story* atau *feature* dari tim lain sih. Jadi kayak tim lain punya cerita dimasukan ke tim ini. Kalau di tim gue pernah itu sih sekali. Coding: Tujuan proyek dapat menjadi tidak jelas.
H7.12	T: Lalu pernah ga sih *requirement*-nya jadi ga jelas? Jadi *goal*-nya sudah oke, tapi pas nentuin *requirement*-nya malah ga jelas. N: Itu lumayan sering tapi kalau disini kita mitigasi-nya jadi awal-awal PM kan yang pertama kali, pasti bingungkan. Awalnya itu gue berguru nya sama PM senior. Tapi ternyata lebih bagus gue berguru sama tim *developer*-nya. Jadi kemarin gue nemuin kalau gue punya *story*, gue ga tahu mau dipecahin *requirement* gimana. Jadi gue *grooming*, tanpa *requirement* jelas. Jadi gue nunjukin *mockup*, bantuin dong apa yang harus dipikirin dibuat disana. Dan itu bagus banget. Tim *developer*-nya jadi ngerasa ngemiliki cerita itu juga. Dan *impact*-

	nya jadi bagus banget. Hal-hal yang ga bisa gue kerjain itu malah bagus banget *impact* nya. Mikirin inilah, kayak kalau *translation*-nya Bahasa Thailand bakalan ga muat. Lebih detail dan terstruktur gitu pikirannya. Coding: Sering terjadi kebutuhan yang kurang jelas, PO mendiskusikan kebutuhan yang kurang jelas bersama *developer*.
H7.13	T: Kalau sebelum nya pas berguru sama PM senior, dampaknya apa ya? N: Itu biasanya dia sudah punya template cerita nya kayak misalnya harus, jadi dia punya *style* cerita sendiri. Kayak misalnya, kalau misalnya *user* gini, maka harus gini, maka ini yang terjadi. Kayak lebih dari sisinya. Bagus sih dapat dari sisi dia. Tapi kalau lihat *board-board* setiap PM pasti *style*-nya beda. Karena itu *styledeveloper*-nya beda. Kayak makanya bagus belajar dari senior PM karena dia punya pengalaman. Tapi memang lebih penting nyesuain sama tim *developer*-nya. Yang gua pelajari itu sih. Setiap tim cara pecahin *story* nya beda-beda. Coding: PO memiliki cara sendiri untuk mengambil keputusan, PO menyesuaikan diri dengan tim *development* yang dia handle.
H7.14	T: Konflik *requirement* antar PM pernah ga? N: Kayaknya pernah. Bukan konflik *requirement* sih, tapi kayak tim A ngerjain halaman A. Nah ternyata tim B juga ngerjain halaman A tapi sisi yang beda. Kayak yang 1 tombol ini, yang satu apa gitu. Nah itu sempet tuh kejadian. Terus akhirnya, jadinya kasian di *developer*-nya sih, karena yang satu timnya migrasi duluan. Tim B nya baru. Gua ga terlalu ngerti sih *technical*-nya gimana mecahinnya. Jadi kita jadi *meeting* antar PO gitu, jadi sekarang kita punya *meeting* 1 minggu sekali untuk mastiin kita ga ada yang nabrak. Dan kita tahu kita saling ngerjain apa. Dan senior PM disini, kayak dia tahu semua ngerjain apa. Jadi kalau *nexttime* ada yang nabrak, dia pasti tahu. Coding: Pernah terjadi konflik kebutuhan, melakukan rapat koordinasi PO untuk menghindari konflik kebutuhan.
H7.15	T: Dulu sering? Atau cuma beberapa kali. N: Selama gue disini sih baru sekali.

	Coding: Jarang terjadi konflik kebutuhan.
H7.16	T: Kemudian pas kan ada biasanya proses ngebobotin User Story, biasanya PM terlibat ga? N: Terlibat. Coding: PM terlibat dalam Sprint Planning.
H7.17	T: PM Terlibat pembobotannya, nngasih bobot atau gimana? N: Enggak, Cuma mastiin mereka ngerti cerita nya. Kayak apa sih ini, termasuk ini gak. Gitu. Coding: PO memastikan tim *developer* mengerti *story* yang akan di-*planning*.
H7.18	T: Tapi disini memang *requirement*nya memang sering berubah ya? N: Sering nya tuh belum ke define. Jadi kayak misalnya sambil jalan, biasanya *developer* nemuin kayak oh ternyata kalau gue ngerubah ini, ini juga kena. Nah gimana tuh, PM nya kan belum mikir kan, jadinya mikir dulu. Tuh bahayanya kalau PM nya lagi banyak *meeting*. Jadi lama jawabnya. Coding: Kebutuhan belum jelas, kesibukan PO menghambat tim *development* untuk mengkonfirmasi kebutuhan yang belum jelas.
H7.19	T: Jadi delay ya? N: Iya. Coding: Pekerjaan menjadi *delay* jika PO sulit dihubungi.
H7.20	T: Dan itu sering kayak gitu? N: Itu lumayan sering sih. Tapi ga pernah sampe kayak efek besar sih. Biasanya kayak hal-hal kecil gitu. Coding: Sering terjadi perubahan kebutuhan.
H7.21	S: Kalau misalkan belum ke-define gitu, *developer*-nya bingung, terus gimana? N: Disini biasanya, untungnya langsung nanya PM dulu, terus kalau disitu kita ga nemuin solusi. Pilihannya A atau B. Gua harus pertimbangin beberapa hal dulu. Jadi gua lari ke senior PM. Keputusannya yang mana. Biasanya dari situ langsung dapet sih keputusannya. Masalahnya adalah kalau gua

	lagi ga bisa mikirin, belum bisa jawab dulu deh. Sehari biasanya. Biasanya PP langsung ngejar gua. Biasanya *developer*nya langsung ngerjar sendiri. Jarang sih PP yang kejar. Coding: Pengembang menanyakan langsung kepada PO mengenai kebutuhan yang kurang jelas, PO berkonsultasi dengan PO yang lebih senior mengenai kebutuhan yang kurang jelas.
H7.22	T: Dari wawancara sebelumnya, ada kolom khusus buat *review* nya PM. Kakak, nge*review* ini kayak ngetest atau cuma lihat saja atau gimana? N: Gua sih ngetest sampe, kita kan punya *acceptance criteria*-nya. Itu gue ngetest dari *acceptance criteria* itu. Kayak misalnya, jika gue pencet tombol ini, maka ini terjadi. Biasanya, nge-*testflow* gagalnya juga. Kayak misalnya, gue kadang-kadang ga nulis sih *flow* gagalnya. Sebenarnya itu salah satu retro kita juga. Semua *flow* gagal ditulis semua dong. Tapi gue belum. Jadi gue ngetest kalau ga ada *connection* gitu-gitu. Coding: PO melakukan *testing* sesuai *acceptance criteria*.
H7.23	T: Itu memang kakak yang bikin mockup sendiri dulu *requirement*-nya apa. N: Kalau desain buat prototype itu yang bikin desainer. Kita sih cuma kayak apa hasilnya yang kita mau. Kayak ide. Tapi biasanya anak-anak sih yang lebih tahu. Ini bagusnya kayak gini karena android pakainya kayak gini. Coding: -
H7.24	T: Kemudian di *daily standup* nya sudah efektif belum? N: Menurut gue sih belum efektif. Seharusnya 15 menit, tapi bisa sampe 30 menit. Gue belum tahu sih gimana caranya, tapi kadang-kadang, kita kan *standupgoal*nya kita saling tahu ada masalah kan dan yang berhubungan kan. Tapi kadang-kadang gue bikin apa, ada urusan yang molor nih. Terus kita ngobrol itu bisa sampe 10 menit lebih. Kalau kayak gitu seharusnya dibawa diluar *standup*. Kalau disini kadang-kadang masih kebahas. Kayak terlalu lama dalam satu topik. Padahal itu seharusnya bisa ga perlu 1 tim dengerin juga gitu. Coding: Daily Scrum belum efektif karena membahas

	hal yang tidak sesuai konteks.
H7.25	T: Itu masih sering terjadi di *daily standup*? N: Masih sering banget terjadi. Karen ague nya juga kepo jadi gue dengerin juga. Coding: Sering terjadi Daily Scrum yang kurang efektif.
H7.26	T: Lalu, kalau dari PM sendiri, kakak itu megang beberapa tim sih? Atau satu saja? N: Satu. Coding: -
H7.27	T: Menurut kakak lebih enak koordinasi sama tim yang anggotanya banyak atau dikit? N: Kalau pengalaman disini karena disini gue timnya lumayan banyak kira kira 8 orang, 6 *developer*, 2 QA, 1 Desainer. Tapi desainernya shared sih semua tim. Sama gue sebelumnya PO di gojek. Itu tim gue, kan dia *outsource*, tapi tim nya kecil sih. Jadi ga bisa dibandingin sih. Enak gede karena banyak suara kan, jadi kayak gue ngasih *story*, mereka banyak ngasih pertimbangan lebih jelas. Kalau enaknya kecil apa ya? Ga deh, enakan gede deh. Tapi ga terlalu gede sampe 20 orang sih. Tapi pas sih segini 8 orang. Coding: Semakin banyak anggota tim Scrum akan memperbanyak saran yang dapat diberikan kepada PO.
H7.28	T: Dan disini ada Business Analyst nya juga ga? N: Ada. Jadi kalau kita data dari database kita punya 2 Business Analyst. Tapi data yang dari Google Analytics, masih gue yang megang. Jadi kalau gue nyari data kayak drop off, atau kayak berapa orang kalau tombol itu masih PM sendiri yang nyari. Pengen sih, *hire*Data Analyst. Kayak nya lagi mau minta sih. Coding: Ada Data Analyst.
H7.29	T: Kemudian, untuk komunikasi antar anggota tim, gimana menurut kakak di HappyFresh? N: Menurut ku bagus banget. Kayak gua ga pernah nemuin tim yang komunikasinya sebagus ini. Kayak ya gue ga mau jelekin perusahaan lain, tapi kayak ini bagus banget. Dari *developer*, desainer sama PM ga ada hierarki sama sekali. Kalau pernah ngerasa, kenapa lu milih nya A sih padahal B lebih bagus, ya

	bakalan debat, sampe ternyata B lebih bagus baru kita pilih B. Kayak banyak pertimbangan gitu. Jadi kayak pas lagi jalan, *story*-nya, sudah kayak gini *fix*. Terus *developer* nemuin sesuatu. Biasanya kalau diperusahaan lain itu ya sudah diam saja. Nanti bakalan jadi maslah. Kalau disini enggak, mereka pasti ngomong. Jadi bisa berubah disini. Jadi *happy* banget disini. Coding: Komunikasi di tim Scrum sudah baik, *developer* pro aktif dalam menyampaikan pendapat.
H7.30	T: Kalau masalah *skill* ngomong nya nih. *Skill* komuniasi setiap anggota tim. Ada ga yang malah kurang bisa komuniasi, terus akhirnya *ruined* the team. N: Kalau di tim gue sih untungnya enggak ada. Tapi sempat ada gue dengar ada satu yang suka, seringnya ngeluh banget, tapi enggak mau ngasih solusi, jadi kayak kenapa sih gini. Kayak oh, sok tahu banget PM nya tapi ga mau ngasih solusi. Itu sih lumayan menganggu. Tapi orangnya sudah keluar dari sini. Coding: *Skill* komunikasi dalam tim Scrum sudah baik.
H7.31	T: Kalau dampaknya ada orang kayak gitu gimana? N: Kalau PM nya sih gue ngelihatnya dia super stress banget sih. Pertama PM orang Indonesia cuma gue, yang lain orang luar. Jadi dia sudah outsider, tapi 1 orang itu makin bikin dia ngerasa *outsider* lagi. Tapi itu karena orang nya yang sudah ga *happy* saja sih. Jadi dia keluar malah bagus. Enggak, maksudnya bukan gua ga ada *pesonal* problem sama dia. Cuma dia yang bikin suasana tim jadi ga enak. Coding: Pengembang yang memiliki kemampuan komunikasi yang baik akan mempengaruhi kinerja PO.
H7.32	T: Terus masalah dokumentasi produk nih, ada gak dokumentasi produk jadi? N: Harusnya ada. Seidealnya ada. Tapi gue, jadi PM kita ada 3 kan. Gue, Joe sama Jinny. Nah si Jinny ini orangnya rapi banget. Pokok nya dia suka organize lah. Itu dia yang paling sering dokumentasi. Kalau gue sama joe ga pernah dokumentasi. Harusnya sih ada. Ini lagi coba dibantu. Jadi kemarin kita lagi coba bikin, biar setiap kali kita *release*, *flow* nya kita bikin baru. Jadi minta bantuan desainer nya. Desainer intern gitu. Itu buruk sih.

	Coding: Ada PO yang merasa malas untuk membuat dokumentasi.
H7.33	T: Dampaknya kalau ga ada dokumen apa sih? N: Dampaknya misalnya gue pengen *tracking*, kadang kadang gue lupa kayak, jadi si PP itu sudah bikin dokumen. Jadi *story* apa saja yang *release*. Kadang-kadang butuh dari sisi kenapanya. Kenapa kita milih A ketimbang B. Nah itu ga ada kan, jadi pas lihat ke belakang, ga inget kenapa kita pilih ini. Sama dari sisi ini lain, si *stakeholder* lain. Misalnya operation, kita buat *feature* yang berhubungan dengan dia, si PP kan buatnya, *story* ini ABCD, sedangkan kita kan bikin cara buat *feature* ini. Karena ga bikin, jadi operationnya nanya ke kita lagi jadi harus jelasin lagi. Kekurangannya ngabisin waktu di belakangnya sih. Jadi kayak kalau operationnya nanya sekali ga apa apa. Tapi kalau dia lupa dan nanya lagi, gua harus jelasin lagi gitu. Coding: Tidak adanya dokumentasi membuat PO mudah lupa mengenai detail kebutuhan yang dibuat, tidak adanya dokumentasi membuat PO harus berkali-kali menjelaskan kepada pihak yang ingin mengetahui mengenai produk yang dikembangkan.

No	**Nama: Ivan Afandi** **Jabatan: Lead UI/UX Designer** **Scrum Role: Development Team**
H8.1	T: Kakak sudah berapa lama kerja di sini? N: Kayak nya setahun 2 bulan deh. Coding: -
H8.2	T: Dan kakak sebagai desainer, ikut juga dalam proses-proses di Scrum? N: Iya. Sebenarnya kalau dibilang lebih spesifik sih. Karena gue sendiri masuk ke tim produk. Jadi, memang bakalan selalu mau ga mau bakalan terlibat dalam proses Scrum karena itu yang diterapin sama tim tech kita, dev kita. Coding: Desainer terlibat dalam proses Scrum.
H8.3	T: Terus, kan kita mau identifikasi risiko kan. Nah, kalau menurut kakak sendiri, risiko dari pakai Scrum apa sih? Terutama dari pandangan seorang desainer.

	N: Risiko nya lebih ke ini kali ya. Kalau misalkan kita merubah sesuatu atau kayak ngasih *update* pas *daily standup* kayak gitu, biasanya pembahasannya jadi lama. Jadi yang ada malah, sebenarnya kita cuma ngupdate, tapi kalau sebagai desainer, pasti semua orang pengen lihat *visual*-nya gitu. Jadinya panjang dan itu kayak ga ideal gitu. Coding: Desainer merasa pembahasan dalam Daily Scrum kurang efektif.
H8.4	T: Memang biasanya *daily standup* nya berapa lama? N: Harus nya sih 10 menit sampe 15 menit paling lama kali ya. Kadang dulu itu sempat *daily standup* 30 menit. Tapi itu pun sebenarnya, jadi dulu si *product* sendiri, kita ga ngikut *standup* nya tech karena kita ngerasa *standup* nya itu beda. Kayak kita itu yang diupdate kayak visual dan ada pembahasannya. Ternyata malah jadi boomerang sama kita, akhirnya kita gabung sama tech. Jadi kita ngupdate nya secara verbal saja. Coding: Waktu Daily Scrum sudah baik, Daily Scrum penting untuk diikuti desainer.
H8.5	S: Berarti kalau misalkan ada *daily standup* yang lama waktunya kurang gimana gitu ya. Mending ngomong *pesonal* saja ya N: Iya, kayak Scrum masternya bilang kita bahas setelah *standup*. Coding: Membahas permasalahan teknis setelah Daily Scrum.
H8.6	T: Lalu, kalau dari desain, sering ga sih disajak *meeting-meeting* diluar *meeting*Scrum? N: Eh, banyak sih. Coding: Banyak *meeting* diluar Scrum.
H8.7	T: Menurut kakak, itu mengganggu kerjaan ga sih? N: Kita dikasih kebebasan sih, mau ikut *meeting* nya atau enggak. Karena *meeting-meeting* itu sebenarnya *meeting-meeting* yang belum masuk ke proses desain tinggi gitu. Masih kayak kembangan dari si UX nya lebih jauh. Ya dibebasin sih ikut atau enggak. Dan *so far* ga ngeganggu. Coding: *Meeting* diluar Scrum tidak mengganggu desainer, desainer bebas memilih untuk ikut atau tidaknya kedalam *meeting* diluar Scrum.

H8.8	T: Lalu, kalau retro nya, ikut ga? N: Ikut. Coding: Desainer terlibat dalam Sprint Retrospective.
H8.9	T: Menurut kakak, retro disini gimana? Bermanfaat ga? N: Bermanfaat banget dan bakal kepakai banget. Coding: Retrospective bermanfaat bagi desainer.
H8.10	T: Rutin ga dilakukan? N: Rutin. Coding: Sprint Retrospective rutin dijalankan.
H8.11	S: Kakak kan sudah 1 tahun lebih disini. Kan kerja nya secara tim. Pernah ga sih kayak ada masalah pribadi gitu dari anggota tim Scrum kakak sendiri. Yang ganggu ke kerjaan. N: *So far* enggak sih. Mungkin karena pakai Scrum juga ya sih. Jadi komunikasi nya lancar. Coding: Tidak ada masalah pribadi dalam Scrum, Scrum membantu mempelancar komunikasi.
H8.12	T: Justru Scrum yang menuntut kita buat *speak up* ya. N: Ya. Coding: Scrum membantu untuk melancarkan komunikasi.
H8.13	T: Tujuan proyek kurang jelas? N: Lebih ke kalau gua lebih tergantung *stakeholder* nya kali ya, atau tergantung PM. Kalau disini malah semuanya *clear* banget. Tapi gua pernah ngalamin pakai Scrum juga di *company* sebelumnya, *goal* nya malah ga jelas. Tergantung orang nya lah. Coding: Tujuan proyek jelas, jelas atau tidaknya tujuan proyek bergantung pada PO yang bersangkutan.
H8.14	S: Kalau disini jelas berarti ya. N: Iya. Coding: Tujuan proyek jelas.
H8.15	T: Kalau *requirement*nya, disainer itu buat *sticky note* ga sih?

	N: Ga ada, tapi kita pakai *Trello*. Coding: Menggunakan *virtual*Scrum Board sebagai pengganti *sticky note*.
H8.16	T: Kakak, pernah ga ngerasa *requirement*-nya ga jelas? N: Pernah beberapa kali. Maksudnya bukan *goal* ya, base nya kurang jelas. Paling ngantisipasi nya langsung ngomong ke orang yang megang *project* itu. Coding: Kebutuhan kurang jelas, mengkonfirmasi kebutuhan yang kurang jelas kepada PO.
H8.17	T: Berarti memang orang nya yang kasih kurang jelas. Itu PM ya yang kasih? N: Iya PM. Coding: -
H8.18	T: Dan dampak nya jadi, apakah takes more *time* atau gimana? N: Itu tergantung *level* senioritas desain dan pengalaman kerja nya kali ya. Kalau junior, dia bakal kerjain dulu yang dia tahu dari itu, baru di lempar. Kalau senior biasanya lebih, tanya dulu. Karena sebagai desainer, lu dituntut ditanya sama *developer*, pas di-merge jadi gimana. Padahal banyak yang beda jadi bentrok. Coding: Level senioritas akan mempengaruhi keputusan yang diambil ketika kebutuhan kurang jelas.
H8.19	T: Ini sering ga terjadi kayak gini? N: Baru kemarin. Coding: Jarang terjadi kebutuhan yang kurang jelas.
H8.20	S: Kalau terjadi kayak gitu, kakak ngatasin nya gimana? N: Biasanya ujung-ujungnya langsung ngomomngin ke *developer* sih. Jadi masing-masing *lead* kayak koordinasi gitu. Sekarang pun *lead*-nya sendiri ada *meeting* buat sinkronisasi gitu, antara IOS dan android atau satu *feature* dengan yang lain. Kalau kemarin, kita quick solution-nya kita kasih visualnya kalau di-*merge* gimana. Sebagai desainer cuma bisa gitu.

	Coding: Melakukan konfirmasi mengenai kebutuhan yang kurang jelas.
H8.21	T: Kalau masalah, kan di awal Sprint itu ada pembobotan masing-masing *story*. Ikut ga proses gitu? N: Grooming ya, ikut. Coding: Desainer ikut dalam proses *planning*.
H8.22	T: Pernah ga sih, atau kalau di desainer, di *developer* kan biasanya ada *dependency*, kalau di desain ada kayak gitu? N: Kayaknya enggak ada. Coding: Tidak ada *dependency* di desain.
H8.23	T: Kalau tadi kan kita ngomong masalah *requirement* ga jelas nih, kalau perubahan *requirement* pernah ga? N: Ada pernah. Coding: Terjadi perubahan kebutuhan didalam Scrum.
H8.24	T: Itu kenapa bisa berubah? N: Mungkin karena, *complexity*, jadi mungkin pas *planning* atau grooming, ternyata ga segampang yang di-*estimate*, jadinya *requirement*nya , jadi sebenarnya nentuin secara *technical*, tapi visualnya ngena. Coding: Proyek yang kompleks menyebabkan kebutuhan bisa berubah, kurang matangnya *planning* membuat kebutuhan dapat berubah.
H8.25	T: Dan dampaknya terjadi perubahan *requirement* itu? N: Iya harus nge-*update* perbaiki lagi. Coding: Pengembang harus mengubah sesuai dengan perubahan yang diminta.
H8.26	T: Sering ga sih? N: Jarang. Coding: Jarang terjadi perubahan kebutuhan.
H8.27	T: Dan kalau desain ini ada *unit testing* nya juga kan ya? N: User *testing* ada. Coding: Ada *user testing* untuk desainer.

H8.28	T: Yang nge-*test* siapa? N: Biasanya PM sama UX Researcher kita. Coding: User *testing* dilakukan oleh PO dan UX Researcher.
H8.29	T: Lalu kalau boleh tahu, desainer nya sendiri ditempatkan di tim desain sendiri dalam 1 tim Scrum, atau desainernya untuk satu tim *development* ada beberapa desainer? N: Dulu sempet gitu sih. Jadi ada 1 desainer 1 tim. Kalau sekarang kayak 1 desainer bisa kerjain 1 sampe 2 tim. Coding: -
H8.30	T: Dan desainer ada *definition of done*-nya ga sih? N: Ada. Coding: Ada *definition of done*.
H8.31	T: Jadi desainer dikatakan done saat apa? N: Pas sudah masuk *build*, dan sudah lewat, jadi kita ada proses *review* by desainer, *testing* QA, sama *review* by PM. Baru *deploy*. Jadi pas sudah *deploy* itu sudah *done*. Coding: Ada *definition of done*.
H8.32	T: Kemudian, kalau kakak sendiri, lebih enak kerja di tim Scrum yang anggota nya banyak atau gimana? N: 12 orang paling banyak kayaknya. Coding: Ada batas dimana *developer* sudah merasa jumlah anggotanya sudah cukup.
H8.33	T: Kalau kebanyakan, ngaruhnya apa? N: Mungkin, jadi terlalu banyak suara, terlalu banyak yang *speak up*. Jadi susah. Coding: Terlalu banyak anggota tim akan membuat terlalu banyak anggota yang berbicara.
H8.34	T: Kalau di HappyFresh sendiri gimana? N: Enggak kok, cuma segitu. Coding: -
H8.35	T: Untuk komunikasi antar desainer dan *developer* gimana?

	N: Membantu banget kalau pakai Scrum. Coding: Scrum membantu komunikasi *developer* dan desainer.
H8.36	T: Ada ga sih yang mungkin ada anggota yang kemampuan komunikasinya kurang, terus mengganggu proses *development*? N: Ada Coding: Ada anggota yang memiliki kemampuan komunikasi yang kurang.
H8.37	T: Kira-kira dampaknya apa ya? N: Dampaknya jadinya, saat ada *changes*, pas pengerjaan, atau pas Sprint, *developer* jadi susah buat nanya ke dia atau cara komunikasi sama dia gimana. Dan dia ga punya *self ownership*. Coding: Sulit untuk mengkonfirmasi pekerjaan dengan orang yang memiliki kemampuan komunikasi yang rendah.
H8.38	S: Berarti kalau gitu cuma di *cover* sama *developer* lain saja kalau ada kayak gitu? N: Jadinya PM yang di kejar terus. Bahkan nanya tentang desain ke PM. Coding: PO harus siap ditanya-tanya oleh orang yang sulit berkomunikasi di dalam tim.
H8.39	T: Kalau desain ada dokumentasi ga? N: Ada. Coding: Ada dokumentasi untuk desain.
H8.40	T: Itu gimana? N: Kita bilangnya kayak *pattern library* dan atomic desain. Pattern library kita pakai buat *mobileapp* atau *mobile* web. Kalau atomic desain memang khusus buat *web*. Coding: Dokumentasi desain berisikan *pattern library* dan *atomic*.
H8.41	T: Kalau antar tim, kakak kan bisa dapet tugas dari banyak tim ya, koordinasinya gimana sih? N: Koordinasinya sama si PM nya. Dan pas *grooming*. Coding: Desainer berkoordinasi dengan PO pada saat Backlog Grooming.

H8.42	T: Kalau sama QA desainer perlu koordinasi ga? N: Perlu. Coding: QA perlu berkordinasi dengan desainer.
H8.43	T: Koordinasi masalah apa? N: Jadi ada beberapa hal yang QA ga terlalu paham. Kayak *interface* nya sebenar nya sudah benar. Tapi mereka ga ngerti. Masalah visual gitu. Coding: QA perlu memahami *interface* untuk melakukan *testing*.
H8.44	T: Kalau koordinasinya baik? N: Baik. Coding: Koordinasi QA dan desainer baik.

LAMPIRAN V

DAFTAR SELURUH KODE

Kode	Coding
K1.1	*Deadlineengineer/developer.*
K1.2	Style penggunaan Scrum masing-masing berbeda.
K1.3	Proyek tidak dapat diestimasi di awal Sprint.
K1.5	Komitmen Pengembang dalam Sprint
K1.5	*Shortdeadline*
K1.5	Perubahan *velocity* di tengah Sprint
K1.6	Adaptif (kerjaan lain bisa masuk ditengah Sprint)
K1.6	komitmen *developer* dapat berubah
K1.7	Prioritas PBI berdasarkan keinginan.
K1.7	Penentuan Product Backlog berdasarkan KPI
K1.8	PO merumuskan PBI
K1.9	Tujuan proyek cukup jelas
K1.9	Proyek yang tidak terlalu besar membuat tujuan lebih jelas
K1.10	Tujuan proyek jelas.
K1.11	Requirement jelas
K1.11	inisiatif meng-*explorerequirement*
K1.12	Requirement kurang jelas di awal Sprint
K1.12	PO membuat definisi produk yang jelas
K1.13	Tandai *story* yang sudah tidak relevan
K1.14	*Lowprioritystory* tidak di-*deliver*
K1.14	Story yang tidak selesai dilanjutkan ke Sprint selanjutnya
K1.15	Konflik *requirement* tidak terjadi pada perusahaan dengan 1 PO
K1.16	Story *dependency*
K1.16	Salah prioritas *story*
K1.16	*impact* untuk *user* kurang baik
K1.17	Perubahan detail *requirement*
K1.18	Planning yang kurang detail
K1.18	Kesalahan *userexperience*
K1.19	Perubahan detail *requirement* berdampak positif
K1.20	Sering terjadi perubahan detail *requirement*
K1.21	Detail desain dapat dinegosiasi
K1.22	*Technical Debt* masuk kedalam *story*
K1.23	Terlalu banyak *bug* dan Technical Debt
K1.23	Moral *engineer* turun karena banyak Technical Debt
K1.24	Konflik Pair Programming
K1.24	Kesiapan Pair Programming
K1.24	Tipe pekerja yang cocok Pair Programming
K1.25	Ketidakcocokan Pair Programming memperlambat *development*
K1.26	Jarang Pair Programming

K1.27	Testing berdasarkan *story*
K1.28	Story tidak perlu terlalu jelas
K1.28	Tim *solid* sudah *self-organize*
K1.29	Tidak ada *dedicated* Scrum Master
K1.29	*Captain*Pengembang pengganti Scrum Master
K1.29	Daily Standup menjadi ajang diskusi hal teknis
K1.29	Stakeholder tidak berkepentingan tidak mendapat *benefit* pada daily *stand up*
K1.30	Daily Scrum jarang tidak efektif
K1.31	Kehabisan PBI dalam Scrum
K1.32	Proses Scrum berbeda antar tim
K1.33	Perbedaan Scrum antar perusahaan
K1.34	Tidak ada Definition of Done tertulis
K1.35	Tidak ada pengaruh antara *velocity* awal dengan hasil akhir
K1.36	Story harus berdampak kepada *user*
K1.37	Pair Programming ketika jumlah anggota tim Scrum banyak
K1.38	Jumlah anggota berbanding lurus dengan jumlah *communication channel*-nya
K1.39	Anggota tim Scrum masih kecil
K1.40	Split tim Scrum
K1.41	Tidak ada Business Analyst
K1.42	Business analyst diperlukan saat perusahaan sudah mau merabah ke bisnis
K1.43	Komposisi tim yang kurang tepat
K1.43	Komunikasi tidak efektif
K1.44	Integrasi di awal
K1.45	*Skill* komunikasi penting
K1.46	Tidak ada dokumentasi produk
K1.47	Pembentukan tim tergantung pada proyek yang akan dijalankan
K1.48	QA terlibat dalam *planning*
K1.49	PO mendelegasikan orang lain sebagai *proxy* untuk berhadapan dengan *developer*
K2.1	Scrum melindungi *developer* dari PO
K2.1	Sprint *planning* tidak matang
K2.2	Scrum melindungi *developer*
K2.3	Scrum tidak digunakan karena tuntutan investor
K2.4	Penambahan *storybugs* saat Sprint berlangsung
K2.5	*Meeting* diluar Scrum mengganggu fokus *developer*
K2.6	*Meeting* diminta oleh pihak manajemen
K2.6	*meeting* seharusnya dipagi atau sore hari
K2.7	Pengembang menjadi kehilangan konsentrasi akibat dari *meeting*
K2.8	Penerapan Scrum yang tidak sesuai lagi
K2.9	Membahas masalah too much *meeting* di *Retrospective*
K2.10	Sprint Retrospective tidak dilaksanakan dengan rutin
K2.10	Scrum tidak dijalankan dengan baik tanpa Scrum

	master khusus
K2.11	Pengembang tidak fokus
K2.11	Ketidak-adaan Scrum master
K2.12	Ada masalah pribadi dalam Scrum
K2.12	Individu harus memiliki kesadaran diri
K2.13	Menyelesaikan masalah pribadi dengan berkomunikasi empat mata
K2.13	Mengungkapkan permasalahaan di *Retrospective*
K2.13	Membicarakan masalah dengan Scrum Master
K2.14	Tujuan selalu dijelaskan pada *planning*
K2.15	Visi misi belum cukup untuk memperjelas tujuan proyek
K2.15	Pengembang tidak mengerti tujuan proyek
K2.16	Kurang jelasnya tujuan proyek
K2.16	Hasil yang tidak sesuai ekspektasi PO
K2.17	Jarang terjadi tujuan proyek tidak jelas
K2.18	Mendiskusikan tujuan proyek pada saat Sprint Planning
K2.19	Requirement cukup jelas
K2.21	PO menentukan prioritas PBI
K2.22	Belum pernah terjadi perubahan *requirement*
K2.23	Perubahan UI produk dimasukan ke *story* pada Sprint selanjutnya
K2.24	Melakukan *refactoring* di Technical Debt
K2.25	*Technical Debt* tidak dimasukan kedalam Scrum
K2.26	Melakukan *meeting* untuk *refactoring*
K2.27	Ada ketidakcocokan saat Pair Programming
K2.27	Pair Programming berdasarkan *codeagreement*
K2.28	Ada permasalahan *pesonal* saat Pair Programming
K2.29	Mengatasi ketidakcocokan dari *interview* kerja
K2.30	Jarang terjadi ketidakcocokan antar *developer*
K2.31	Unit *testing* tidak menggunakan dokumen
K2.31	Unit *testing* berdasarkan *cleanarchitecture*
K2.31	Ada dokumentasi API
K2.32	Standup *meeting* efektif
K2.32	Melanjutkan bahasan teknis diluar *standupmeeting*.
K2.33	Penyusunan tim Scrum berdasarkan bidang keahlian
K2.34	Implementasi Scrum dalam perusahaan tergantung pada Scrum Master
K2.35	Story yang *dependency* dimasukan ke *next*Sprint
K2.36	Scrum *board* yang terpisah
K2.37	Dependency menyebabkan *developer* mengerjakan *story* yang tidak *dependent* terlebih dahulu
K2.38	Sering terjadi *dependency*
K2.39	Adanya *definition of done*
K2.40	Burn down chart
K2.40	Velocity awal tidak tepat
K2.41	Kurangnya pengalaman *developer* dalam mengukur *velocity*

K2.42	Velocity harus sesuai dengan kemampuan *developer*
K2.42	Velocity rendah maka *task* sedikit
K2.42	Velocity tinggi maka *task* banyak
K2.43	Velocity awal tidak tepat
K2.44	*Customervalue* dan *cleancode* sama pentingnya
K2.45	Terlalu banyak anggota tim Scrum mengganggu kinerja *prorgammer*
K2.46	Terlalu banyak anggota tim Scrum mengganggu komunikasi dan membuat produk aplikasi lebih banyak *bug*
K2.48	Masih kekurangan sumber daya manusia
K2.49	Tidak adanya Business Analyst
K2.50	Perlu adanya Business Analyst
K2.51	Business analyst diperlukan jika perusahaan mau mendapatkan profit
K2.52	Komunikasi *developer* dilakukan pada *codereview*
K2.53	Code *review* cukup untuk melakukan komunikasi *developer*
K2.54	*Skill* komunikasi mempengaruhi proses *development*
K2.55	Komunikasi yang kurang tidak memberikan dampak yang berarti
K2.56	Sering ada *developer* yang memiliki kemampuan komunikasi yang kurang
K2.57	Dokumentasi untuk membantu *maintenance*
K2.57	Dokumentasi belum diperlukan apabila jumlah *developer* sedikit
K2.58	Koordinasi masalah komunikasi melalui Scrum master
K2.59	Koordinasi dibantu oleh *engineering* manager yang mempunyai technical knowledge
K2.60	Kurangnya kolaborasi antar *developer* dan QA
K2.61	Kurangnya keterlibatan QA dalam Scrum
K2.62	Kurangnya keterlibatan QA dalam Scrum memicu aplikasi yang *bug*gy
K2.63	Unit *testing* tanpa QA
K2.64	Kolaborasi QA dan *developer* kurang baik
K2.65	PO sudah handal
K3.1	Pengembang tidak bisa memilih *story*
K3.3	Pengembang harus siap diganggu untuk komunikasi saat Sprint
K3.4	Banyak *meeting* diluar Scrum
K3.5	*Meeting* diluar Scrum tidak mengganggu karena durasinya singkat
K3.7	Retrospective tidak jalan
K3.8	Retrospective itu efektif
K3.8	Pembahasan *Retrospective* monoton
K3.9	Tidak adanya *Retrospective* tidak terlalu berdampak pada anggota tim Scrum
K3.10	Jarang terjadi masalah pribadi dalam Scrum
K3.12	Tujuan proyek jelas
K3.13	Requirement jelas

K3.13	Pengembang sering lupa *requirement*
K3.14	Pengembang bertanya ke PO apabila lupa *requirement*-nya
K3.15	Prioritas *requirement* tidak memadai
K3.16	Story penting tidak selesai
K3.17	Jarang terjadi salah estimasi
K3.18	Jarang terjadi perubahan *requirement*
K3.20	Estimasi ulang untuk perubahan *requirement*
K3.20	Melakukan drop *story*
K3.21	Perubahan *requirement* menyebabkan *story* tidak selesai
K3.22	Pernah melakukan Technical Debt dalam Scrum
K3.23	Pengembang memasukan Technical Debt kedalam *story*
K3.24	Pair Programming memiliki standard yang harus diikuti
K3.25	Terjadi ketidakcocokan dalam Pair Programming
K3.26	Tidak ada dampak ketidakcocokan Pair Programming pada pekerjaan
K3.27	Dapat memilih pasangan Pair Programming
K3.28	Kadang-kadang *developer* dipasangkan dengan orang yang tidak cocok dalam Pair Programming
K3.31	Standup *meeting* efektif
K3.32	Standup *meeting* terlalu lama
K3.33	Membahas hal teknis adalah penyebab Daily Scrum terlalu lama
K3.34	Pengembang tidak fokus pada Daily Scrum
K3.35	Proses Scrum di tiap tim sama
K3.36	Definition of done jelas
K3.37	Salah estimasi *velocity* diawal
K3.38	Pengembang hanya merasa sebagai eksekutor
K3.40	Business analyst diperlukan saat sudah mau monetize
K3.41	Tidak ada masalah komunikasi antar anggota tim
K3.42	*Skill* komunikasi yang kurang dari *developer* tidak mempengaruhi *development*
K3.43	Perlu adanya dokumentasi *product*
K3.44	Pengembang malas membuat dokumentasi.
K3.45	Tidak adanya dokumentasi membuat proses inisiasi anggota baru menjadi lebih lama
K3.46	Diperlukan proses *onboarding* untuk karyawan baru
K3.47	Sering terjadi *dependency*
K3.48	Dependency disebabkan karena kurangnya komunikasi antar tim
K3.50	PO sudah handal
K3.51	PO belum bisa mengukur secara kuantitatif
K3.52	Kurang handalnya PO menyebabkan *developer* tidak bisa mengukur dengan tepat efek dari penambahan *feature*
K4.2	Ada *meeting* tambahan diluar Scrum
K4.3	*Meeting* diluar Scrum tidak mengganggu pekerjaan
K4.4	Sprint retro berjalan baik

K4.7	Tidak ada masalah pribadi
K4.8	Tujuan proyek jelas
K4.9	Kadang terjadi *requirement* desain yang tidak jelas
K4.10	Requirement desain kurang jelas karena desain bersifat subjektif
K4.11	Melakukan eskplore dan membuat beberapa desain sebagai alternative
K4.12	Revisi desain
K4.14	Pernah terjadi perubahan *requirement*, perubahan *requirement* berdasarkan statistiK
K4.16	Jarang merubah desain
K4.17	Desainer jarang mengikuti Daily Scrum
K4.18	Daily Scrum belum terlalu efektif untuk desainer
K4.19	Ketidakadaan progress dari desainer tidak menjadi masalah pada Daily Scrum
K4.20	Sering terjadi Daily Scrum yang kurang efektif
K4.21	Menyampaikan progress desain di Daily Scrum
K4.22	Ada *definition of done* yang jelas
K4.25	Perlu adanya businesss analyst
K4.27	Tidak ada masalah komunikasi
K4.28	Penyesuaian diri terhadap karakter anggota tim
K4.29	Kurangnya kemampuan komunikasi seorang anggota menuntut anggota lain untuk lebih aktif
K4.30	Tidak ada *dedicated* PO
K5.1	Pengembang tidak mengetahui *requirement* secara keseluruhan
K5.1	Pengembang mengerjakan *story* yang tidak sesuai dengan yang ditetapkan
K5.2	Kesalahan estimasi terjadi karena belum mengetahui *velocity* tim
K5.3	Tidak bisa mengukur *velocity* saat pertama menggunakan Scrum
K5.4	Sulit untuk mengestimasi *velocity* untuk pertama kalinya
K5.5	Ada *meeting* diluar Scrum
K5.6	*Meeting* diluar Scrum mengganggu
K5.6	Terlalu banyak *meeting* mengurangi produktivitas *developer*
K5.7	Sprint Retrospective bermanfaat
K5.8	Sprint retorspektif tidak rutin dilaksanakan
K5.9	Sprint Retrospective yang tidak berjalan akan membuat keluh kesah *developer* tidak dapat tersampaikan
K5.10	Terjadi perdebatan antar *developer* mengenai metode yang digunakan
K5.11	Voting untuk menentukan metode
K5.12	Tujuan proyek jelas
K5.13	PBI sudah jelas
K5.14	Requirement sudah lengkap
K5.14	Melakukan pembahasan detail pada pembuatan *task*

K5.15	Tidak terjadi salah prioritas PBI
K5.15	Prioritas PBI bisa diubah dengan menyampaikan pada PO
K5.16	Perubahan *requirement* dimasukan kedalam Sprint
K5.17	Perubahan *requirement* disebabkan oleh permintaan PO
K5.18	Sering terjadi perubahan *requirement*
K5.20	Pernah melakukan Technical Debt
K5.21	*Technical Debt* sebagai upaya *maintenancecode*
K5.22	Tidak ada masalah kecocokan pada saat Pair Programming
K5.23	Unit *testing* dilakukan oleh *developer*
K5.23	Dokumentasi dianggap melelahkan
K5.24	Meminimalisir dampak tidak adanya dokumentasi dengan TDD
K5.25	Standup *meeting* sudah efektif
K5.26	Proses Scrum di tiap tim sama
K5.27	Ada *definition of done* yang jelas
K5.28	Kesalahaan estimasi dalam menentukan *velocity*
K5.29	Pernah terjadi kesalahan estimasi
K5.30	Semakin banyak anggota tim yang ahli maka pekerjaan akan cepat selesai
K5.31	Belum membutuhkan bantuan Business Analyst
K5.32	Komunikasi dalam Scrum baik
K5.32	Komunikasi dibantu oleh teknologi Slack
K5.33	Ada dokumentasi untuk API
K5.34	Menyediakan dokumentasi API terlebih dahulu untuk digunakan oleh tim yang memerlukan
K5.35	Tidak ada masalah kolaborasi QA dan *developer*
K5.36	PO sudah handal
T1.1	Task tidak selesai
T1.1	Task yang tidak selesai dilanjutkan pada Sprint selanjutnya
T1.2	Pengembang sakit
T1.2	Ada *task* lain yang prioritasnya lebih tinggi
T1.3	Dilanjutkan ke Sprint selanjutnya
T1.4	Task yang tidak selesai dipengaruhi oleh proses bisnis dari suatu tim
T1.5	Rapat diluar Scrum untuk *feature* baru
T1.6	Rapat diluar Scrum mengganggu
T1.7	Rapat diluar Scrum membuat pekerjaan tertunda
T1.8	Rapat diluar Scrum jarang dilakukan
T1.9	Ada Sprint retro di akhir Sprint
T1.10	Sprint retro berguna
T1.11	Retro membantu *introvert* untuk menyampaikan pendapat
T1.12	Ada masalah pribadi antara QA dan *developer*
T1.13	Penyesuaian terhadap sistem Scrum
T1.15	Tujuan proyek kurang jelas pada Scrum
T1.16	Perbedaan hasil dengan ekspektasi tim bisnis
T1.17	Kurang koordinasi antar tim

T1.18	Kurang koordinasi terjadi di awal-awal penggunaan Scrum
T1.19	PBI yang dibuat PO sudah jelas
T1.20	Ada beberapa PO dalam satu perusahaan
T1.21	Pencegahan konflik *requirement* dengan membuat modul berbeda untuk masing-masing PO
T1.22	Sering terjadi *dependency*
T1.23	Perubahan yang mendadak menyebabkan *dependency*
T1.24	Membagi tugas *developer* yang tidak memiliki *task* banyak untuk menyelesaikan permasalahan depdency
T1.25	Permintaan tim untuk mengubah *requirement*
T1.26	Testcase sebagai dokumen *testing*
T1.27	Daily Scrum kurang efektif bila tidak ada update progress
T1.28	Scrum master mencegah *developer* membahas teknis di Daily Scrum
T1.29	Proses Scrum menyesuaikan dengan tim yang bersangkutan
T1.30	Ada *definition of done* yang jelas
T1.31	QA tidak mengukur *velocity*
T1.32	Lebih mudah mengatur anggota tim dengan jumlah sedikit
T1.33	Tidak ada masalah komunikasi
T1.33	Inisiatif *developer* untuk mengkonfirmasi perubahan
T1.34	Tidak ada masalah *skill* komunikasi
T1.35	Tidak memungkinkan terjadinya overlap *requirement*
T1.36	Test dilakukan saat *developer* sudah selesai
T1.37	PO sudah handal
T2.1	Story pada Sprint sering mundur ke Sprint selanjutnya
T2.1	Sering ada tambahan pekerjaan diluar Sprint
T2.2	Dependency antar tim
T2.3	Inisiatif *developer* untuk mengerjakan apa yang bisa dikerjakan selama terjadi *dependency*
T2.4	Jarang terjadi *dependency*
T2.5	Pekerjaan yang tak terduga datang dari pihak manajemen
T2.6	Jarang terjadi tambahan pekerjaan yang tidak terduga
T2.7	Retro tidak selalu dilaksanakan
T2.8	Sprint Retrospective bermanfaat
T2.8	Sprint Retrospective tidak rutin dijalankan
T2.9	Tidak ada dampak signifikan dengan tidak diadakannya *Retrospective*
T2.10	Tidak ada masalah pribadi didalam Scrum
T2.11	Tujuan proyek cukup jelas
T2.12	Requirement kadang-kadang kurang jelas
T2.13	Diskusi tambahan untuk memperjelas *requirement*
T2.14	Mengutus team *lead* untuk melakukan *meeting* agar tidak terjadi konflik *requirement* antar tim
T2.15	Terjadi *dependency*

T2.16	Dependency terjadi ketika ada *task* yang dipecah menjadi lebih kecil
T2.17	Jarang terjadi *taskdependency*
T2.18	Belum terjadi perubahan *requirement*
T2.20	Sharing sebagai pengganti techical debt
T2.21	Terjadi ketidakcocokan Pair Programming
T2.21	Mencoba untuk lebih mengerti teman Pair Programming
T2.22	Mencoba untuk melakukan diskusi
T2.23	Jarang terjadi ketidakcocokan Pair Programming
T2.24	Pengembang melakukan *unit testing*
T2.25	Tidak memerlukan panduan test *developer*
T2.26	Daily Scrum sudah efektif
T2.27	Waktu Daily Scrum sudah baik
T2.28	Membahas masalah teknis pada Daily Scrum
T2.29	Daily Scrum dapat membantu *developer* lain untuk saling memberi masukan
T2.30	Jarang membahas teknis di Daily Scrum
T2.32	Ada *definition of done*
T2.33	Pengembang tidak mengukur *velocity*
T2.34	Lebih efektif bekerja dengan anggota tim Scrum yang sedikit
T2.37	Tidak ada masalah komunikasi
T2.38	*Skill* komunikasi anggota tim sudah baik
T2.39	Tidak terdapat dokumentasi produk
T2.40	Perlu adanya dokumentasi produk
T2.41	Dependency terjadi pada pembuatan API
T2.42	Perbedaan prioritas antar tim memicu *dependency*
T2.43	QA dan *developer* berkoordinasi dengan baik
T2.44	Testing dilakukan per-*feature*
T2.44	Ada *testing* keseluruhan *feature*
T3.1	Perubahan *requirement* mudah terjadi di perusahaan internet
T3.1	Perubahan *requirement* datang dari pihak manajemen
T3.1	Harus membuat keputusan yang cepat di waktu yang terbatas
T3.1	Tidak dapat melakukan proper *development* di waktu yang terbatas
T3.2	PO bertugas melakukan negosiasi kepada *development* team untuk memasukan *task* tambahan
T3.3	PBI dibuat oleh tim
T3.3	Harus memahami kapasitas kerja *developer*
T3.4	Velocity diusahakan stabil
T3.5	Pengembang terlibat dalam prioritas *storybugs*
T3.5	PO lebih terlibat dalam prioritas *story* baru
T3.6	Tujuan proyek jelas karena PO berusaha menjelaskan tujuan proyek
T3.7	Perubahan *requirement* terjadi di awal
T3.7	Membuat rule untuk perubahan *requirement*
T3.8	Membuat modul masing-masing untuk tiap PO

T3.9	Semakin lama menggunakan Scrum, tim akan menjadi lebih handal dalam menentukan prioritas
T3.10	Over*estimate* pekerjaan di awal menggunakan Scrum
T3.11	Sering terjadi over*estimatestory*
T3.12	PO mengikuti proses Daily Scrum
T3.12	Jika PO tidak dapat hadir pada Daily Scrum, maka ia mendelegasikan tugasnya ke Scrum master
T3.13	Definition of done jelas
T3.14	Designer ikut Daily Scrum untuk memantau pekerjaan *developer*
T3.15	Terjadi over*estimate* terhadap kerjaan
T3.16	Over*estimate* terjadi di awal-awal menggunakan Scrum
T3.17	Komunikasi di Daily Scrum harus fokus
T3.17	Pemanfaatan teknologi untuk komunikasi
T3.17	Menjaga hubungan baik tim di luar jam kerja
T3.18	Melakukan rolling anggota tim
T3.18	Tim yang berisiskan *introvert* akan membuat suasana kerja tidak menyenangkan
T3.19	Team mencari sendiri anggotanya saat reqruitment
T3.20	Ada dokumentasi produk berupa PRD
T3.20	ada PO yang malas membuat dokumentasi
T3.20	Dokumentasi dianggap membuang waktu
T3.21	Mengkoordinir lebih dari 1 tim Scrum membuat PO tidak bekerja maksimal
T4.1	Belum terbiasa dengan *framework*Scrum
T4.1	*self-organize* dari *development* team
T4.2	Training untuk anggota baru mengenai Scrum
T4.3	Training dilakukan oleh tim Scrum sendiri
T4.4	Memberikan gambaran kepada anggota yang belum paham supaya bisa *self-organize*
T4.5	*Meeting* di luar Scrum tergantung pada tim masing-masing
T4.7	Retrospective tidak selalu dijalankan setiap 1 *cycle*Sprint
T4.8	Retrospective tidak selalu dijalankan setiap 1 *cycle*Sprint
T4.9	Tidak ada masalah pribadi antar tim Scrum
T4.10	Tujuan proyek pada Scrum jelas
T4.10	Dapat terjadi perubahan prioritas
T4.11	Perubahan prioritas menyebabkan pekerjaan tertunda
T4.12	Perubahan prioritas jarang terjadi
T4.14	Perubahan harus tetap diikuti
T4.15	Requirement selalu jelas
T4.16	Pair Programming dilakukan dalam Scrum
T4.17	Tidak pernah ada masalah ketidakcocokan saat Pair Programming
T4.18	Adanya *unit testing*
T4.19	Daily Scrum sudah efektif
T4.20	Waktu Daily Scrum tergantung jumlah tim

T4.21	Daily Scrum membahas hal yang efektif
T4.22	Membahas teknis yang penting pada Daily Scrum
T4.23	Anggota tidak masalah dengan waktu Daily Scrum yang lama
T4.24	Definition of done jelas
T4.25	QA membuat *test case*
T4.26	Pembobotan PBI tergantung pada fitur yang dikerjakan
T4.27	Pembobotan *story* tidak konstan sesuai dengan kesulitan *feature*
T4.29	Tim Scrum tidak boleh memiliki anggota terlalu banyak
T4.29	Tim terlalu banyak akan susah diatur
T4.30	Tim yang besar harus di pantau saat Daily Scrum
T4.31	Jumlah anggota tim tidak bergantung pada proyek
T4.32	Penggunaan teknologi untuk membantu komunikasi dalam Scrum
T4.33	Kurangnya *skill* komunikasi akan menghambat pekerjaan
T4.34	Ada anggota yang sulit untuk mengikuti kesepakatan awal
T4.35	QA dan *developer* berkolaborasi dengan baik
T4.36	Setiap tim memiliki 1 PO yang mengurusinya
T4.37	Tidak pernah terjadi overlap pekerjaan antar tim
T4.37	Pernah terjadi *dependency* antar tim
T4.38	Ada dokumentasi produk dalam bentuk wiki
T5.1	Desain tidak maksimal jika menggunakan Scrum
T5.2	Desainer menggunakan Scrum didalam Scrum untuk dapat menjalan kan tugasnya
T5.3	Sering *meeting* informal di luar Scrum
T5.4	*Meeting* informal diluar Scrum tidak mengganggu pekerjaan
T5.5	Tim desain memiliki proses Sprint Retrospective pada Scrumnya
T5.6	Sprint Retrospective rutin dilakukan
T5.7	Retrospective digunakan untuk belajar dari kesalahan
T5.8	Terjadi permasalahan pribadi antar anggota
T5.8	Saling introspeksi diri
T5.9	Jarang terjadi permasalahan pribadi di dalam Scrum
T5.10	*Skill* komunikasi yang kurang mempengaruhi *development*
T5.10	Kurangnya *skill* komunikasi dapat menyebabkan *miscommunication*
T5.11	Tujuan proyek jelas
T5.12	Requirement kurang jelas karena kurangnya komunikasi
T5.13	Kurang jelasnya *requirement* menyebabkan pekerjaan tidak maksimal
T5.14	Menyisipkan *requirement* tambahan yang kecil ke dalam Sprint
T5.14	Mengerjakan requiremen tambahan yang besar di Sprint selanjutnya
T5.15	Jarang terjadi perubahan *requirement*

T5.16	Tim Scrum desain diurus oleh beberapa PO
T5.17	Tidak ada konflik *requirement* antar PO
T5.17	pembagian tugas PO berdasarkan modul
T5.18	Tim susah menentukan prioritas untuk mengerjakan permintaan dari PO
T5.19	Jarang terjadi *dependency* di tim desain
T5.20	Sering terjadi perubahan *requirement*
T5.20	Planning yang kurang matang menyebabkan ekspektasi yang tinggi
T5.21	Daily Scrum sudah efektif
T5.22	Daily Scrum digunakan untuk membahas masalah teknis
T5.23	Daily Scrum yang lama akan membuat tim menjadi tahu akan masalah dan siap menghadapi masalah
T5.24	Waktu Daily Scrum efektif
T5.25	Penambahan waktu pada Daily Scrum tidak menjadi masalah
T5.26	Ada *definition of done* yang jelas
T5.27	Penentuan bobot dilakukan berdasarkan kesepakatan tim
T5.29	Semakin sedikit anggota tim maka *development* akan semakin efektif
T5.30	Susah untuk mengatur pembagian kerja pada anggota tim yang banyak
T5.31	Pemecahan tim menjadi lebih kecil ketika sudah terlalu banyak anggotanya
T5.32	Ada dokumen untuk desain dalam bentuk weekly report
T5.34	Tidak ada overlap *requirement* dalam tim desain
T5.35	Sprint desainer dan *developer* dilakukan bersamaan
T5.36	Tim desain belum memiliki PO yang berdedikasi khusus
T6.1	Mudah terjadi *miscommunication*
T6.1	Mudah terjadi perubahan
T6.1	Perubahan membuat pekerjaan menjadi lebih lama
T6.2	Perubahan sering terjadi
T6.3	Menggunakan Daily Scrum untuk meminimalisir *miscommunication*
T6.4	Scrum master lebih dekat dengan *developer*
T6.4	PO lebih dekat dengan pihak manajemen
T6.6	Senior *developer* akan menjadi trainer bagi anggota baru
T6.7	Waktu Scrum bergantung pada kebutuhan dan kondisi
T6.8	Tujuan proyek jelas karena sudah ditentukan setiap Sprint Review
T6.9	Requirement jelas karena dibahas setelah menentukan *goal*
T6.10	PO mempunyai modul masing-masing
T6.11	Prioritas *requirement* dibuat oleh PO
T6.12	Planning *requirement* kurang matang dan kurang sumber
T6.13	Jarang melakukan Pair Programming
T6.13	Menggunakan teknologi untuk menggantikan Pair Programming

T6.14	Scrum master memantau proses Daily Scrum dan Sprint yang berlangsung
T6.15	Scrum Master mengajak anggota tim untuk melakukan Daily Scrum
T6.16	Proses Scrum di tiap tim sama
T6.17	Definition of done berbeda tiap tim
T6.17	Definition of done jelas
T6.18	Pembobotan PBI dilakukan oleh *developer* yang akan mengerjakannya
T6.19	Keefektifan jumlah anggota tergantung pada kondisi proyek
T6.21	Menggunakan Daily Scrum untuk mengantisipasi *miscommunication*
T6.22	Menggunakan *Retrospective* untuk mengatasi *miscommunication*
T6.23	*Skill* komunikasi memperngaruhi proses *development*
T6.24	Jarang terjadi *miscommunication*
T6.25	Kurangnya dokumentasi produk
T6.25	Sulit untuk mengajarkan anggota baru tanpa menggunakan dokumentasi
T6.26	Kurangnya dokumentasi menyebabkan proses pembelajaran anggota baru terhadap produk yang ada menjadi lama
T6.27	Terjadi *dependency* antar tim Scrum
T6.28	Terjadinya *dependency* disebabkan karena urutan proses bisnisnya
T6.29	Planning yang lebih matang dari tim produk
T7.1	Pekerjaan terkesan seperti tidak selesai
T7.1	Tidak jelasnya arah pengerjaan
T7.1	Tidak jelasnya metode pengukuran bobot atau prioritas
T7.1	Belum fasih dalam menerapkan Scrum
T7.2	Karakter individu mempengaruhi pekerjaan yang tidak selesai
T7.3	Kurangnya kepercayaan untuk meminta bantuan *role* yang ada di Scrum
T7.4	Sulit dalam melakukan proses scoring
T7.5	Scoring menggunakan kertas membuang-buang waktu
T7.6	Mengerjakan sesuai prioritas tanpa melakukan scoring
T7.7	Ada *meeting* diluar Scrum
T7.8	*Meeting* diluar Scrum menggaggu waktu *developer*
T7.8	*Meeting* di luar Scrum membantu memperjelas proyek kedepan
T7.9	Retrospective dihilangkan karena memakan waktu
T7.10	Tidak ada masalah pribadi di dalam tim Scrum
T7.10	Melakukan diskusi informal untuk menyelesaikan masalah
T7.11	Tujuan proyek sudah jelas
T7.12	Requirement tidak jelas
T7.13	Requirement tidak jelas karena hanya diberikan ide

	umumnya saja
T7.14	Pengembang harus menentukan sendiri apa yang harus dilakukan berdasarkan general idea yang diberikan
T7.15	Terjadi konflik *requirement* antar PO
T7.16	Tidak terlalu sering terjadi konflik *requirement*
T7.17	PO menentukan prioritas yang ingin dikerjakan
T7.18	Kesalahan prioritas terjadi di awal menggunakan Scrum
T7.19	Selalu terjadi perubahan *requirement*
T7.19	Pernecanaan kurang matang
T7.19	Kurangnya pengetahuan produk
T7.20	Perubahan *requirement* membuat *developer* harus menambah waktu kerja mereka
T7.21	Sering terjadi perubahan *requirement*
T7.22	Perubahan *requirement* harus tetap dikerjakan
T7.23	Tidak melakukan Pair Programming
T7.23	Pair Programming dianggap kurang menghargai privasi
T7.24	Ada *unit testing*
T7.24	Testing lebih efektif dilakukan oleh QA
T7.25	Tidak ada dokumen *testing* untuk *developer*
T7.26	Perlu adanya dokumentasi
T7.27	Daily Scrum sulit untuk efektif karena banyak anggotanya
T7.28	Terlalu banyak anggota menyebabkan Daily Scrum semakin kurang efektif
T7.29	Hal yang dibahas pada Daily Scrum sudah sesuai
T7.30	Daily Scrum menjadi lebih lama di tim Scrum yang jumlah anggotanya banyak
T7.31	Waktu Daily Scrum sudah baik
T7.33	Proses Scrum antar tim berbeda
T7.34	Scrum *board* digunakan sebagai media
T7.35	Ada *definition of done* yang jelas
T7.36	Ada Business Analyst dalam Scrum
T7.37	Business analyst terkadang bertentangan dengan PO
T7.38	Komunikasi berjalan dengan baik
T7.39	Tidak ada masalah dengan *skill* komunikasi
T7.40	Sudah ada dokumentasi produk akhir
T7.41	Perlu adanya koordinasi antar tim Scrum
T7.42	Kolaborasi *engineer* dengan QA sudah baik
T7.43	Satu tim dipegang oleh satu Product Owner khusus
H1.1	Story pada Scrum yang kurang jelas
H1.1	Perubahan *story*
H1.2	Sprint Retrospective membantu untuk mengevaluasi Sprint yang telah dikerjakan
H1.3	Menggunakan Scrum *board* virtual untuk memperjelas *story*
H1.4	Scrum *board* virtual untuk membantu komunikasi mengenai *story*
H1.5	Ada *meeting* diluar Scrum

H1.5	*Meeting* dengan orang bisnis
H1.6	*Meeting* dengan pihak bisnis berguna bagi *developer*
H1.7	*Meeting* diluar Scrum tidak mengganggu *developer*
H1.8	Sprint Retrospective berguna
H1.8	Pelaksanaan Sprint Retrospective tergantung pada Scrum master
H1.9	Pengembang tidak dapat menyampaikan keluh-kesah tanpa Sprint Retrospective
H1.10	Tidak ada masalah pribadi dalam Scrum
H1.10	Terjadi perbedaan pendapat mengenai arsitektur yang akan digunakan
H1.12	Tujuan proyek jelas untuk produk yang dibangun dari awal
H1.13	Requirement kurang detail mengarah pada perubahan *requirement*
H1.14	Jarang terjadi perubahan *requirement* karena kurang detail
H1.15	Jarang terjadi perubahan *requirement* karena kurang detail
H1.16	Tidak bisa mengerjakan *story* yang belum jelas
H1.16	Mengubah prioritas *story* yang belum jelas
H1.17	Product owner disebut sebagai *product* manager
H1.18	Setiap tim diurus oleh satu Product Owner
H1.19	Tidak ada overlap *requirement*
H1.19	Terjadi *dependency*
H1.20	Kurangnya koordinasi antar tim menyebabkan *dependency*
H1.21	Waktu untuk menyelesaikan suatu *feature* menjadi semakin lama sebagai dampak dari *dependency*
H1.22	Jarang terjadi *dependency*
H1.23	Dependency diatasi dengan melakukan diskusi terlebih dahulu
H1.24	*Technical Debt* dilakukan diluar Sprint
H1.25	*Technical Debt* dilakukan saat dibutuhkan
H1.26	Ada Pair Programming didalam Scrum
H1.27	Tidak ada konflik dalam Pair Programming
H1.27	Pair Programming dilakukan oleh seorang senior dan seorang junior
H1.28	Pair Programming bertujuan untuk meningkatkan kualitas *code*
H1.29	Jarang dilakukan Pair Programming
H1.30	Ada *unit testing*
H1.31	Testcase sebagai dokumen *testing*
H1.32	Daily Scrum masih kurang efektif
H1.33	Daily Scrum tidak efektif dalam segi pembahasan
H1.34	Proses Scrum antar tim berbeda
H1.34	Scrum *board* antar tim berbeda
H1.35	Definition of done jelas
H1.36	Pembobotan backlog menggunakan skala angka (Planning poker)

H1.37	Ada *velocity*
H1.37	Sprint Retrospective digunakan untuk mengevaluasi *velocity*
H1.38	Anggota tim merasa nyaman bekerja di tim yang anggotanya tidak terlalu banyak
H1.39	Ada Business Analyst
H1.40	Ada anggota yang mengalami masalah komunikasi
H1.41	Tidak ada dokumentasi produk
H1.42	Ada beberapa tim yang menggunakan dokumentasi
H1.43	Koordinasi tim dilakukan dengan melakukan *meetinglead*
H1.44	QA melakukan *testing* saat *story* masuk ke kolom *testing*
H1.45	Ada *dedicated* PM
H2.1	Proses dalam Scrum berlangsung dengan cepat
H2.1	Sering muncul hambatan yang tidak terdeteksi di-*planning*
H2.2	Story yang tidak selesai dilanjutkan ke Sprint selanjutnya
H2.3	*Meeting* diluar Scrum tidak menjadi masalah apabila terjadwal dengan baik dan durasinya tidak lebih dari 1 jam
H2.4	Waktu rapat dapat menjadi lama karena banyaknya hal yang harus dibahas
H2.5	Sering terjadi perubahan jadwal *meeting*
H2.6	Perubahan jadwal *meeting* tidak mengganggu konsentrasi *developer*
H2.6	perubahan jadwal *meeting* mengganggu jadwal *developer*
H2.7	Sprint Retrospective tidak berjalan dengan rutin
H2.8	Sprint Retrospective berdampak positif pada tim Scrum
H2.8	Sprint retrosepktif menjadi ajang instropeksi
H2.9	Tidak ada masalah pribadi antar anggota tim
H2.10	Belum ada masalah pribadi yang dibawa ke pekerjaan
H2.11	Tujuan proyek kurang jelas
H2.11	PO kurang mendeskripsikan tujuan proyek kepada *developer*
H2.12	Diperlukan alasan yang kuat untuk tiap backlog yang dikerjakan
H2.13	Beberapa *developer* kurang bisa mendapatkan maksud dari proyek yang disampaikan PO
H2.14	Sering terjadi perubahan *requirement*
H2.15	Tidak ada ovelap *requirement*
H2.15	Terjadi bentrok pada *code* yang dibuat
H2.16	Terjadi Dependency
H2.17	Dependency terjadi karena kurang matangnya analisis saat *planning*
H2.18	Jarang terjadi *dependency*
H2.19	Tidak ada Technical Debt
H2.20	Pair Programming dilakukan saat awal anggota baru

H2.21	Pair Programming digunakan bagi anggota baru untuk beradaptasi
H2.22	Tidak ada masalah ketidakcocokan dalam Pair Programming
H2.23	Pengembang melakukan *unit testing*
H2.24	Pengembang membuat dokumen *testing* sendiri berdasarkan *story*
H2.25	Waktu Daily Scrum setiap tim berbeda
H2.26	Daily Scrum sudah efektif
H2.27	Waktu Daily Scrum sudah efektif
H2.28	Proses Scrum antar tim sama
H2.29	Seorang Scrum master dapat meng-handle beberapa tim sekaligus
H2.30	Sudah ada *definition of done* yang jelas
H2.31	Terjadi over*estimate* terhadap *story*
H2.32	Tidak ada masalah komunikasi dalam Scrum
H2.34	Dokumen produk akhir diurus oleh PO dan Scrum master
H2.35	Test QA dapat dilakukan jika *developer* sudah selesai mengerjakannya
H2.36	Ada *dedicated* PM untuk tiap tim
H3.1	Praktisi Scrum belum memahami *mindset*Agile
H3.2	Melakukan on *board*ing untuk menyamakan persepsi orang baru terhadap *mindset*Agile di perusahaan
H3.2	menggunakan *Retrospective* untuk memperbaiki kesalahan *mindset* anggota
H3.3	Penerapan Scrum di tiap perusahaan berbeda
H3.3	Scrum hanyalah sebuah *framework* yang fleksibel
H3.4	Menambah nilai manfaat dari Scrum agar *developer* tertarik untuk berpartisipasi
H3.5	Product Owner disebut sebagai Product Manager
H3.6	Story yang tidak selesai dilanjutkan ke Sprint selanjutnya
H3.7	Perpanjangan waktu Sprint akan mengganggu kinerja Sprint tim lain
H3.8	Tujuan proyek jelas selama PO memiliki visi yang jelas
H3.9	Tujuan proyek sudah jelas
H3.10	Product Owner sudah menjelaskan poin Why dari suatu tujuan proyek
H3.11	Requirement sering kurang jelas, perlu adanya inisiatif dari *developer* untuk memperjelas *requirement* yang kurang jelas
H3.12	Requirement yang kurang jelas membuat *developer* harus menanyakan ulang kepada PO
H3.13	Tidak terjadi konflik *requirement*
H3.14	Sulit untuk menyatukan *requirement* yang konflik
H3.15	Pernah terjadi over*estimate* terhadap *story*
H3.16	Jarang terjadi miss-estimation
H3.17	Ada perubahan *requirement* di tengah Sprint
H3.19	Selalu ada perubahan di Agile*organization*

H3.20	Pengembang harus siap menghadapi perubahan *requirement*
H3.21	Story memiliki kriteria sebelum bisa di-*develop*
H3.21	*story* dapat berubah di Agile*organization*
H3.22	Development team dapat melakukan Daily Scrum tanpa Scrum master
H3.23	Daily Scrum sudah efektif
H3.24	Ada *definition of done*
H3.25	Tiap tim memiliki Definition of done yang sama
H3.26	Ada *definition of done* yang jelas
H3.27	Jumlah anggota tim sudah sesuai dengan ketentuan pada Scrum guides
H3.28	Kefektifan jumlah tim dalam Scrum relative
H3.29	Business analyst sudah merupakan tugas dari PO
H3.30	Role dalam Scrum sudah cukup untuk menjalankan *development*
H3.31	Scrum master bertanggung jawab atas masalah komunikasi yang terjadi
H3.32	Scrum memaksa anggotanya untuk dapat berkomunikasi dengan baik
H3.33	Melakukan sesi sync up antar tim untuk membahas *code* convention
H3.34	Terdapat 1 PM yang meng-handle beberapa tim
H3.35	Perlu adanya PO khusus untuk tiap tim
H3.36	PM yang meng-handle beberapa tim akan memperlambat kinerja tim
H4.2	Ada risiko yang harus dihadapi saat menggunakan Scrum
H4.3	Requirement kurang detail berpotensi menimbulkan *bugs*
H4.4	Pengembang harus menentukan sendiri detail yang harus dikerjakan
H4.5	Kurang detail menyebabkan QA harus bertanya ke *developer* dan PO
H4.6	Menanyakan kembali *requirement* yang kurang jelas kepada PO dan *developer*
H4.7	Ada *meeting* informal diluar Scrum
H4.8	Sprint Retrospective selalu rutin dijalankan
H4.9	Sprint Retrospective bermanfaat untuk mengevaluasi proses *development*
H4.10	Tidak ada masalah pribadi di dalam Scrum
H4.11	Tujuan proyek kurang jelas di Scrum
H4.12	Tujuan proyek kurang jelas
H4.13	PO memberikan tujuan yang terlalu umum
H4.14	Requirement terlalu umum
H4.15	Bugs baru diketahui ketika sudah sampai *production*
H4.16	Sering terjadi *bugs* yang tidak teridentifikasi sebelumnya
H4.17	Pengembang akan menanyakan *requirement* yang kurang jelas kepada PO

H4.19	QA terlibat dalam proses pembobotan
H4.19	QA menyerahkan pembobotan kepada *engineer*
H4.20	Sering terjadi perubahan *requirement*
H4.22	Sering terjadi perubahan *requirement*
H4.23	Perubahan *requirement* menyenbabkan status *task* kembali ke in progress
H4.24	Dokumen *testing* berupa *testcase*
H4.25	QA selalu ikut dalam kegiatan Daily Scrum
H4.26	Daily Scrum sudah efektif
H4.27	Daily Scrum dilakukan tidak lebih dari 15 menit
H4.29	Proses Scrum setiap tim sama
H4.30	Sudah ada *definition of done* yang jelas
H4.32	Jumlah tim Scrum sudah sesuai dengan yang disarankan Scrum guide
H4.34	Komunikasi di tim Scrum lancar
H4.35	Kemampuan komunikasi tidak mempengaruhi proses *development*
H4.36	Tidak ada masalah *skill* komunikasi
H4.37	Belum ada dokumentasi untuk produk akhir
H4.38	Perlu adanya dokumentasi produk akhir
H4.39	Ada atau tidaknya dokumentasi tergantung dari kinerja PO
H4.40	Tidak adanya dokumentasi membuat QA kesulitan melakukan *regression test*
H4.42	Koordinasi dilakukan saat *regression test* untuk QA
H4.43	Test dilakukan setiap *story* selesai
H4.44	Ada PO yang meng-handle setiap tim
H5.1	Under*estimate* terhadap *story* yang dikerjakan
H5.2	Scrum disalah artikan sebagai mini Waterfall
H5.3	Terkadang Scrum dipandang sebagai mini Waterfall
H5.5	Ada kondisi yang menyebabkan Agile tidak dapat diimplementasikan
H5.5	mini Waterfall dan Agile itu berbeda
H5.6	Scrum dapat diimplementasikan dengan baik seiring dengan semakin lamanya organisasi sudah menerapkan Scrum
H5.7	Ada *meeting* diluar Scrum
H5.8	*Meeting* third party sebagai *meeting* diluar Scrum
H5.9	*Meeting* diluar Scrum mengganggu *developer*
H5.9	tidak semua anggota Scrum harus mengikuti *meeting* diluar Scrum
H5.10	Ada *meeting* khusus untuk para team *lead*
H5.11	Sprint Retrospective rutin dijalankan
H5.12	Penting atau tidaknya Sprint Retrospective itu bergantung dengan *developer* itu sendiri
H5.13	Tidak ada masalah pribadi dalam Scrum
H5.14	Masalah komunikasi membuat tujuan proyek menjadi kurang jelas
H5.15	Tujuan proyek yang kurang jelas berdampak ke produktivitas

H5.16	Beberapa kali terjadi tujuan proyek kurang jelas
H5.17	Meningkatkan komunikasi antara PM dan *developer* agar tujuan proyek menjadi jelas
H5.19	Terjadi konflik *requirement*
H5.20	Sering terjadi perubahan *requirement*
H5.21	Jarang terjadi overestimation terhadap *story*
H5.22	Sering terjadi underestimation terhadap *story*
H5.23	Jarang terjadi perubahan *story* di tengah Sprint
H5.25	Tidak ada Technical Debt
H5.26	Kurang sumber daya manusia menyebabkan Pair Programming tidak dilakukan
H5.28	Backend melakukan *unit testing*
H5.28	Frontend tidak melakukan *unit testing*
H5.29	Daily Scrum menjadi ajang membahas masalah teknis
H5.30	Proses Scrum antar tim sama
H5.31	Ada *definition of done* yang jelas
H5.32	PM melakukan test sesuai *acceptance criteria*
H5.34	Efektif atau tidaknya Scrum tidak bergantung pada jumlah anggota tim
H5.35	Komunikasi antar anggota tim Scrum bagus
H5.36	Ada orang yang memiliki kemampuan komunikasi yang kurang
H5.37	Kurangnya kemampuan komunikasi anggota tim membuat proses *development* menjadi lambat
H5.38	Sudah ada dokumentasi produk akhir
H5.39	Pengembang kurang memerlukan dokumentasi produk
H5.40	Setiap tim memiliki satu PO
H5.41	Platform sync up sebagai upaya koordinasi antar tim *developer*
H5.42	QA melakukan *testing* saat *task* sudah masuk ke kolom *testing*
H6.1	Pengembang baru mengalami under*estimate* terhadap *story*
H6.2	Pengembang baru harus didampingi oleh senior dalam melakukan *planning*
H6.3	Ada *meeting* diluar Scrum
H6.4	Banyak *meeting* diluar Scrum
H6.5	Terlalu banyak *meeting* akan mengganggu kinerja *developer*
H6.6	Sprint Retrospective rutin dijalankan
H6.7	Sprint Retrospective penting untuk melakukan evaluasi proses *development*
H6.8	Tidak ada masalah pribadi antar anggota tim Scrum
H6.9	Tujuan proyek jelas
H6.10	Requirement dapat menjadi tidak jelas
H6.11	PO membuat *story* yang kurang jelas
H6.12	PO menentukan *story* bersama dengan *developer*
H6.13	Story yang kurang jelas membuat proses *development* menjadi lama
H6.15	Pernah terjadi konflik *requirement* karena PO

H6.16	Pengembang melakukan pembobotan *story*
H6.17	Over*estimatestory* berbahaya terhadap *development*
H6.18	Pernah terjadi over*estimatestory* dan under*estimatestory*
H6.19	Over*estimate* terjadi karena *developer* perlu memikirikan hal hal tak terduga yang mungkin terjadi
H6.20	Terjadi perubahan *requirement* di tengah Sprint
H6.21	Perubahan *requirement* berupa tambahan fungsi yang harus dikerjakan
H6.22	Sering terjadi perubahan *requirement*
H6.23	Perubahan *requirement* mengganggu proses *development*
H6.24	Perubahan *requirement* harus tetap dikerjakan oleh *developer*
H6.25	Ada Technical Debt
H6.26	*Technical Debt* dilakukan didalam Sprint
H6.27	*Technical Debt* dilakukan untuk memperbaiki *bug*
H6.28	Pernah melakukan Pair Programming.
H6.29	Tidak ada masalah ketidakcocokan saat melakukan Pair Programming
H6.30	Tim *backend* melakukan *unit testing*
H6.30	Tim frontend melakukan UI *testing*
H6.31	Pengembang melakukan test sendiri terhadap UI produk
H6.32	Pengembang membuat *testcase* untuk melakukan *unit testing* dan UI *testing*
H6.33	Daily Scrum sudah efektif
H6.33	Waktu Daily Scrum terlalu siang
H6.35	Waktu Daily Scrum bergantung pada apa yang dibahas
H6.36	Daily Scrum sudah membahas hal yang sesuai konteks
H6.37	Ada *definition of done* yang jelas
H6.38	Pengembang lebih suka bekerja di tim yang sedikit
H6.40	Komunikasi antar anggota tim lancar
H6.41	Tidak ada anggota yang memiliki kemampuan komunikasi yang kurang
H6.42	Ada dokumentasi produk akhir
H6.42	Dokumentasi produk akhir dibuat oleh Scrum master
H6.43	Tidak ada koordinasi khusus antar tim
H6.44	QA melakukan *testing* ketika *task* sudah masuk kolom *testing*
H6.45	Satu tim di-handle oleh satu PO
H7.1	PO disebut sebagai PM
H7.6	Waktu PO untuk membuat *story* lama
H7.6	Waktu Scrum yang terbatas membuat Product Owner kesulitan untuk menentukan PBI untuk Sprint selanjutnya
H7.6	Turunnya moral *developer* apabila mengejakan pekerjaan yang kurang bernilai
H7.6	ketidakhadiran Product Owner pada Daily Scrum akan membuat *miscommunication*
H7.7	Tim secara aktif berkomunikasi dengan PO
H7.8	Product Owner bertugas menentukan Product Backlog

H7.9	Prioritas PBI ditentukan berdasarkan *impact* yang diberikan tiap *story*
H7.9	Memisahkan *board* untuk *bugs* dan permintaan negara dengan *board* untuk *productdevelopment*
H7.10	Tujuan proyek kurang jelas
H7.11	Tujuan proyek dapat menjadi tidak jelas
H7.12	Sering terjadi *requirement* yang kurang jelas
H7.12	PO mendiskusikan *requirement* yang kurang jelas bersama *developer*
H7.13	PO memiliki cara sendiri untuk mengambil keputusan
H7.13	PO menyesuaikan diri dengan tim *development* yang dia handle
H7.14	Pernah terjadi konflik *requirement*
H7.14	melakukan rapat koordinasi PO untuk menghindari konflik *requirement*
H7.15	Jarang terjadi konflik *requirement*
H7.16	PM terlibat dalam Sprint Planning
H7.17	PO memastikan tim *developer* mengerti *story* yang akan di-*planning*
H7.18	Requirement belum jelas
H7.18	kesibukan PO menghambat tim *development* untuk mengkonfirmasi *requirement* yang belum jelas
H7.19	Pekerjaan menjadi delay jika PO sulit dihubungi
H7.20	Sering terjadi perubahan *requirement*
H7.21	Pengembang menanyakan langsung kepada PO mengenai *requirement* yang kurang jelas
H7.21	PO berkonsultasi dengan PO yang lebih senior mengenai *requirement* yang kurang jelas
H7.22	PO melakukan *testing* sesuai *acceptance criteria*
H7.24	Daily Scrum belum efektif karena membahas hal yang tidak sesuai konteks
H7.25	Sering terjadi Daily Scrum yang kurang efektif
H7.27	Semakin banyak anggota tim Scrum akan memperbanyak saran yang dapat diberikan kepada PO
H7.28	Ada Data Analyst
H7.29	Komunikasi di tim Scrum sudah baik
H7.29	Pengembang pro aktif dalam menyampaikan pendapat
H7.30	*Skill* komunikasi dalam tim Scrum sudah baik
H7.31	Pengembang yang memiliki kemampuan komunikasi yang baik akan mempengaruhi kinerja PO
H7.32	Ada PO yang merasa malas untuk membuat dokumentasi
H7.33	Tidak adanya dokumentasi membuat PO mudah lupa mengenai detail *requirement* yang dibuat
H7.33	tidak adanya dokumentasi membuat PO harus berkali-kali menjelaskan kepada pihak yang ingin mengetahui mengenai produk yang dikembangkan
H8.2	Desainer terlibat dalam proses Scrum
H8.3	Desainer merasa pembahasan dalam Daily Scrum kurang efektif

H8.4	Waktu Daily Scrum sudah baik
H8.4	Daily Scrum penting untuk diikuti desainer
H8.5	Membahas permasalahan teknis setelah Daily Scrum
H8.6	Banyak *meeting* diluar Scrum
H8.7	*Meeting* diluar Scrum tidak mengganggu desainer
H8.7	desainer bebas memilih untuk ikut atau tidaknya kedalam *meeting* diluar Scrum
H8.8	Desainer terlibat dalam Sprint Retrospective
H8.9	Retrospective bermanfaat bagi desainer
H8.10	Sprint Retrospective rutin dijalankan
H8.11	Tidak ada masalah pribadi dalam Scrum
H8.11	Scrum membantu mempelancar komunikasi
H8.12	Scrum membantu untuk melancarkan komunikasi
H8.13	Tujuan proyek jelas
H8.13	Jelas atau tidaknya tujuan proyek bergantung pada PO yang bersangkutan
H8.14	Tujuan proyek jelas
H8.15	Menggunakan virtual Scrum *board* sebagai pengganti *sticky note*
H8.16	Requirement kurang jelas
H8.16	Mengkonfirmasi *requirement* yang kurang jelas kepada PO
H8.18	Level senioritas akan mempengaruhi keputusan yang diambil ketika *requirement* kurang jelas
H8.19	Jarang terjadi *requirement* yang kurang jelas
H8.20	Melakukan konfirmasi mengenai *requirement* yang kurang jelas
H8.21	Desainer ikut dalam proses *planning*
H8.22	Tidak ada *dependency* di desain
H8.23	Terjadi perubahan *requirement* didalam Scrum
H8.24	Proyek yang kompleks menyebabkan *requirement* bisa berubah
H8.24	Kurang matangnya *planning* membuat *requirement* dapat berubah
H8.25	Pengembang harus mengubah sesuai dengan perubahan yang diminta
H8.26	Jarang terjadi perubahan *requirement*
H8.27	Ada *user testing* untuk desainer
H8.28	User *testing* dilakukan oleh PO dan UX Researcher
H8.30	Ada *definition of done*
H8.31	Ada *definition of done*
H8.32	Ada batas dimana *developer* sudah merasa jumlah anggotanya sudah cukup
H8.33	Terlalu banyak anggota tim akan membuat terlalu banyak anggota yang berbicara
H8.35	Scrum membantu komunikasi *developer* dan desainer
H8.36	Ada anggota yang memiliki kemampuan komunikasi yang kurang
H8.37	Sulit untuk mengkonfirmasi pekerjaan dengan orang yang memiliki kemampuan komunikasi yang rendah

H8.38	PO harus siap ditanya-tanya oleh orang yang sulit berkomunikasi di dalam tim
H8.39	Ada dokumentasi untuk desain
H8.40	Dokumentasi desain berisikan pattern library dan atomic
H8.41	Desainer berkoordinasi dengan PO pada saat backlog grooming
H8.42	QA perlu berkordinasi dengan desainer
H8.43	QA perlu memahami *interface* untuk melakukan *testing*
H8.44	Koordinasi QA dan desainer baik

LAMPIRAN VI
KUMPULAN KODE UNTUK TIAP RISIKO

Risiko 1: Ketidakefektifan komunikasi

K1.43	Komunikasi tidak efektif
K1.45	*Skill* komunikasi penting
K2.56	Sering ada *developer* yang memiliki kemampuan komunikasi yang kurang
K2.58	Koordinasi masalah komunikasi melalui Scrum master
T4.33	Kurangnya *skill* komunikasi akan menghambat pekerjaan
T5.10	Kurangnya *skill* komunikasi dapat menyebabkan *miscommunication*
T6.23	*Skill* komunikasi memperngaruhi proses *development*
H3.31	Scrum master bertanggung jawab atas masalah komunikasi yang terjadi
H3.32	Scrum memaksa anggotanya untuk dapat berkomunikasi dengan baik
H5.37	Kurangnya kemampuan komunikasi anggota tim membuat proses *development* menjadi lambat
H8.37	Sulit untuk mengkonfirmasi pekerjaan dengan orang yang memiliki kemampuan komunikasi yang rendah

Risiko 2: Sprint Planning kurang Matang

H6.19	Over*estimate* terjadi karena *developer* perlu memikirikan hal hal tak terduga yang mungkin terjadi
H6.18	Pernah terjadi over*estimatestory* dan under*estimatestory*
H6.17	Over*estimatestory* berbahaya terhadap *development*
H5.1	Under*estimate* terhadap *story* yang dikerjakan
H3.16	Jarang terjadi miss-estimation
H3.15	Pernah terjadi over*estimate* terhadap *story*
H2.31	Terjadi over*estimate* terhadap *story*
T3.16	Over*estimate* terjadi di awal-awal menggunakan Scrum
T5.20	Planning yang kurang matang menyebabkan ekspektasi yang tinggi
K2.1	Sprint *planning* tidak matang
H7.16	PM terlibat dalam Sprint Planning
T3.1	Harus membuat keputusan yang cepat di waktu yang terbatas

Risiko 3: Daily Scrum Kurang Efektif

H8.4	Daily Scrum penting untuk diikuti desainer

H8.3	Desainer merasa pembahasan dalam Daily Scrum kurang efektif
H7.24	Daily Scrum belum efektif karena membahas hal yang tidak sesuai konteks
H7.6	ketidakhadiran Product Owner pada Daily Scrum akan membuat *miscommunication*
H6.35	Waktu Daily Scrum bergantung pada apa yang dibahas
H5.29	Daily Scrum menjadi ajang membahas masalah teknis
T7.30	Daily Scrum menjadi lebih lama di tim Scrum yang jumlah anggotanya banyak
T6.15	Scrum Master mengajak anggota tim untuk melakukan Daily Scrum
T2.30	Jarang membahas teknis di Daily Scrum
T2.29	Daily Scrum dapat membantu *developer* lain untuk saling memberi masukan
K3.32	Standup *meeting* terlalu lama
K3.34	Pengembang tidak fokus pada Daily Scrum
K1.29	Stakeholder tidak berkepentingan tidak mendapat *benefit* pada daily *stand up*

Risiko 4: Sprint Retrospective Tidak Rutin Diadakan

H6.7	Sprint Retrospective penting untuk melakukan evaluasi proses *development*
H5.12	Penting atau tidaknya Sprint Retrospective itu bergantung dengan *developer* itu sendiri
H2.7	Sprint Retrospective tidak berjalan dengan rutin
H2.8	Sprint Retrospective berdampak positif pada tim Scrum
H2.8	Sprint retrosepktif menjadi ajang instropeksi
H1.8	Pelaksanaan Sprint Retrospective tergantung pada Scrum master
H1.9	Pengembang tidak dapat menyampaikan keluh-kesah tanpa Sprint Retrospective
H1.2	Sprint Retrospective membantu untuk mengevaluasi Sprint yang telah dikerjakan
T7.9	Retrospective dihilangkan karena memakan waktu
K3.8	Pembahasan *Retrospective* monoton
H1.37	Sprint Retrospective digunakan untuk mengevaluasi *velocity*
T5.7	Retrospective digunakan untuk belajar dari kesalahan
T1.11	Retro membantu *introvert* untuk menyampaikan pendapat
K2.10	Scrum tidak dijalankan dengan baik tanpa Scrum master khusus
K2.11	Ketidak-adaan Scrum master
K2.8	Penerapan Scrum yang tidak sesuai lagi

Risiko 5: Tambahan Pekerjaan di Tengah Sprint

H6.21	Perubahan kebutuhan berupa tambahan fungsi yang harus dikerjakan
T5.14	Menyisipkan kebutuhan tambahan yang kecil ke dalam Sprint
T5.14	Mengerjakan kebutuhan tambahan yang besar di Sprint selanjutnya
T2.6	Jarang terjadi tambahan pekerjaan yang tidak terduga
T2.1	Sering ada tambahan pekerjaan diluar Sprint
T2.5	Pekerjaan yang tak terduga datang dari pihak manajemen

Risiko 6: Adanya Dependency

H2.17	Dependency terjadi karena kurang matangnya analisis saat *planning*
H1.20	Kurangnya koordinasi antar tim menyebabkan *dependency*
H1.21	Waktu untuk menyelesaikan suatu *feature* menjadi semakin lama sebagai dampak dari *dependency*
T6.27	Terjadi *dependency* antar tim Scrum
T6.28	Terjadinya *dependency* disebabkan karena urutan proses bisnisnya
T2.41	Dependency terjadi pada pembuatan API
T2.42	Perbedaan prioritas antar tim memicu *dependency*
T2.16	Dependency terjadi ketika ada *task* yang dipecah menjadi lebih kecil
T1.23	Perubahan yang mendadak menyebabkan *dependency*
K3.48	Dependency disebabkan karena kurangnya komunikasi antar tim
K2.37	Dependency menyebabkan *developer* mengerjakan *story* yang tidak *dependent* terlebih dahulu

Risiko 7: Bekerja dengan tidak maksimal

K1.1	*Deadlineengineer/developer.*
K1.5	Komitmen *developer* dalam Sprint
T5.1	Desain tidak maksimal jika menggunakan Scrum
T1.2	Pengembang sakit
T1.1	Task yang tidak selesai dilanjutkan pada Sprint selanjutnya
T1.1	Task tidak selesai

Risiko 8: Jumlah anggota tim tidak efektif

H8.32	Ada batas dimana *developer* sudah merasa jumlah anggotanya sudah cukup

H7.27	Semakin banyak anggota tim Scrum akan memperbanyak saran yang dapat diberikan kepada PO
H6.38	Pengembang lebih suka bekerja di tim yang sedikit
H5.34	Efektif atau tidaknya Scrum tidak bergantung pada jumlah anggota tim
H1.38	Anggota tim merasa nyaman bekerja di tim yang anggotanya tidak terlalu banyak
T6.19	Keefektifan jumlah anggota tergantung pada kondisi proyek
T5.30	Susah untuk mengatur pembagian kerja pada anggota tim yang banyak
K5.30	Semakin banyak anggota tim yang ahli maka pekerjaan akan cepat selesai
K2.46	Terlalu banyak anggota tim Scrum mengganggu komunikasi dan membuat produk aplikasi lebih banyak *bug*
K2.48	Masih kekurangan sumber daya manusia
K1.38	Jumlah anggota berbanding lurus dengan jumlah *communication channel*-nya

Risiko 9: Tidak adanya Product Owner yang khusus

H7.19	Pekerjaan menjadi delay jika PO sulit dihubungi
T7.3	Kurangnya kepercayaan untuk meminta bantuan *role* yang ada di Scrum
K4.30	Tidak ada *dedicated* PO
K3.52	Kurang handalnya PO menyebabkan *developer* tidak bisa mengukur dengan tepat efek dari penambahan *feature*
K1.29	Tidak ada *dedicated* Scrum Master
K1.31	Kehabisan PBI dalam Scrum
H7.6	Waktu Scrum yang terbatas membuat Product Owner kesulitan untuk menentukan PBI untuk Sprint selanjutnya
H7.6	Waktu PO untuk membuat *story* lama
H7.6	Turunnya moral *developer* apabila mengejakan pekerjaan yang kurang bernilai

Risiko 10: Kurang Keterlibatan QA di dalam proses Scrum

K2.60	Kurangnya kolaborasi antar *developer* dan QA
K2.61	Kurangnya keterlibatan QA dalam Scrum
K2.62	Kurangnya keterlibatan QA dalam Scrum memicu aplikasi yang *buggy*
H8.43	QA perlu memahami *interface* untuk melakukan *testing*

Risiko 11: *Meeting* tambahan yang mengganggu

H8.6	Banyak *meeting* diluar Scrum
H6.5	Terlalu banyak *meeting* akan mengganggu kinerja

	developer
H5.9	tidak semua anggota Scrum harus mengikuti *meeting* diluar Scrum
H4.42	Koordinasi dilakukan saat *regression test* untuk QA
H4.7	Ada *meeting* informal diluar Scrum
H2.5	Sering terjadi perubahan jadwal *meeting*
H2.6	Perubahan jadwal *meeting* tidak mengganggu konsentrasi *developer*
H2.6	perubahan jadwal *meeting* mengganggu jadwal *developer*
H1.6	*Meeting* dengan pihak bisnis berguna bagi *developer*
T1.7	Rapat diluar Scrum membuat pekerjaan tertunda
K5.6	Terlalu banyak *meeting* mengurangi produktivitas *developer*
K2.6	*Meeting* diminta oleh pihak manajemen
K2.6	*meeting* seharusnya dipagi atau sore hari
K2.7	Pengembang menjadi kehilangan konsentrasi akibat dari *meeting*

Risiko 12: Kesalahan Penafsiran Mindset terhadap *mindset*Agile

H5.5	mini Waterfall dan Agile itu berbeda
H5.5	Ada kondisi yang menyebabkan Agile tidak dapat diimplementasikan
H5.3	Terkadang Scrum dipandang sebagai mini Waterfall
H5.2	Scrum disalah artikan sebagai mini Waterfall
H3.4	Menambah nilai manfaat dari Scrum agar *developer* tertarik untuk berpartisipasi
H3.3	Scrum hanyalah sebuah *framework* yang fleksibel
H3.1	Praktisi Scrum belum memahami *mindset*Agile
K3.38	Pengembang hanya merasa sebagai eksekutor

Risiko 13: Kebutuhan yang Tidak Jelas

H8.20	Melakukan konfirmasi mengenai kebutuhan yang kurang jelas
H8.19	Jarang terjadi kebutuhan yang kurang jelas
H8.18	Level senioritas akan mempengaruhi keputusan yang diambil ketika kebutuhan kurang jelas
H8.16	Mengkonfirmasi kebutuhan yang kurang jelas kepada PO
H7.21	PO berkonsultasi dengan PO yang lebih senior mengenai kebutuhan yang kurang jelas
H7.18	kesibukan PO menghambat tim *development* untuk mengkonfirmasi kebutuhan yang belum jelas

H7.12	Sering terjadi kebutuhan yang kurang jelas
H6.13	Story yang kurang jelas membuat proses *development* menjadi lama
H6.11	PO membuat *story* yang kurang jelas
H4.14	kebutuhan terlalu umum
H4.3	kebutuhan kurang detail berpotensi menimbulkan *bugs*
H1.13	kebutuhan kurang detail mengarah pada perubahan kebutuhan
H1.15	Jarang terjadi perubahan kebutuhan karena kurang detail
T7.14	Pengembang harus menentukan sendiri apa yang harus dilakukan berdasarkan general idea yang diberikan
K5.1	Pengembang tidak mengetahui kebutuhan secara keseluruhan

Risiko 14: Tujuan Proyek Tidak Jelas

H8.13	Jelas atau tidaknya tujuan proyek bergantung pada PO yang bersangkutan
H7.11	Tujuan proyek dapat menjadi tidak jelas
H5.15	Tujuan proyek yang kurang jelas berdampak ke produktivitas
H5.14	Masalah komunikasi membuat tujuan proyek menjadi kurang jelas
H4.13	PO memberikan tujuan yang terlalu umum
H2.11	PO kurang mendeskripsikan tujuan proyek kepada *developer*
H2.12	Diperlukan alasan yang kuat untuk tiap backlog yang dikerjakan
H2.13	Beberapa *developer* kurang bisa mendapatkan maksud dari proyek yang disampaikan PO
K2.16	Hasil yang tidak sesuai ekspektasi PO
K2.17	Jarang terjadi tujuan proyek tidak jelas
K1.9	Proyek yang tidak terlalu besar membuat tujuan lebih jelas

Risiko 15: Kurangnya Dokumentasi

K1.46	Tidak ada dokumentasi produk
K2.57	Dokumentasi untuk membantu *maintenance*
K3.43	Perlu adanya dokumentasi *product*
K3.45	Tidak adanya dokumentasi membuat proses inisiasi anggota baru menjadi lebih lama
K5.23	Dokumentasi dianggap melelahkan
T3.20	Ada dokumentasi produk berupa PRD
T3.20	Ada PO yang malas membuat dokumentasi

T3.20	Dokumentasi dianggap membuang waktu
T6.25	Sulit untuk mengajarkan anggota baru tanpa menggunakan dokumentasi
T6.26	Kurangnya dokumentasi menyebabkan proses pembelajaran anggota baru terhadap produk yang ada menjadi lama
H4.39	Ada atau tidaknya dokumentasi tergantung dari kinerja PO
H4.40	Tidak adanya dokumentasi membuat QA kesulitan melakukan *regression test*
H5.39	Pengembang kurang memerlukan dokumentasi produk
H6.42	Dokumentasi produk akhir dibuat oleh Scrum master
H7.33	Tidak adanya dokumentasi membuat PO mudah lupa mengenai detail kebutuhan yang dibuat
H7.33	tidak adanya dokumentasi membuat PO harus berkali-kali menjelaskan kepada pihak yang ingin mengetahui mengenai produk yang dikembangkan

Risiko 16: Ketidakcocokan Pair Programming

K1.24	Konflik Pair Programming
K1.24	Kesiapan Pair Programming
K1.24	Tipe pekerja yang cocok Pair Programming
K1.25	Ketidakcocokan Pair Programming memperlambat *development*
K2.27	Ada ketidakcocokan saat Pair Programming
K2.28	Ada permasalahan *pesonal* saat Pair Programming
K3.28	Kadang-kadang *developer* dipasangkan dengan orang yang tidak cocok dalam Pair Programming
T2.23	Jarang terjadi ketidakcocokan Pair Programming
T7.23	Pair Programming dianggap kurang menghargai privasi
H5.26	Kurang sumber daya manusia menyebabkan Pair Programming tidak dilakukan

Risiko 17: Perubahan kebutuhan yang sangat cepat

H6.24	Perubahan kebutuhan harus tetap dikerjakan oleh *developer*
H6.23	Perubahan kebutuhan mengganggu proses *development*
H6.20	Terjadi perubahan kebutuhan di tengah Sprint
H4.23	Perubahan kebutuhan menyebabkan status *task* kembali ke in progress
H3.21	*story* dapat berubah di Agile*organization*
H3.19	Selalu ada perubahan di Agile*organization*
T7.20	Perubahan kebutuhan membuat *developer* harus menambah waktu kerja mereka

T6.1	Perubahan membuat pekerjaan menjadi lebih lama
T4.10	Dapat terjadi perubahan prioritas
T4.11	Perubahan prioritas menyebabkan pekerjaan tertunda
T3.1	Perubahan kebutuhan mudah terjadi di perusahaan internet
T3.1	Perubahan kebutuhan datang dari pihak manajemen
K3.21	Perubahan kebutuhan menyebabkan *story* tidak selesai

LAMPIRAN VII

RISK REGISTER

Berdasarkan hasil analisis terhadap transkrip wawancara dan interpretasi hasil yang dibantu oleh teori dan hasil penelitian terdahulu, penulis merangkum risiko yang teridentifikasi kedalam suatu risk register sebagai berikut:

No	Risiko	Deskripsi	Kategori	Penyebab	Dampak
1	Ketidakefektifan komunikasi	Proses komunikasi kurang baik yang terjadi karena aspek karakter manusia dan kemampuan komunikasi masing-masing anggota.	Manajemen Proyek	1. Adanya anggota yang *introvert*. 2. Karakter anggota yang sulit mengikuti kesepakatan. 3. Keterbatasan penggunaan Bahasa.	1. Suasana kerja menjadi tidak nyaman. 2. Proses *development* menjadi lama. 3. Memberikan hasil yang tidak sesuai ekspektasi. 4. Perlu adanya perantara dalam berkomunikasi berbeda Bahasa.
2	Sprint *planning* kurang matang	Tim Scrum membuat keputusan perencanaan Sprint yang kurang tepat pada saat melakukan Sprint Planning.	Manajemen Proyek	1. *Overestimate* terhadap User Story. 2. *Underestimate* terhadap User Story. 3. Belum ada	1. Pengembang menjadi tidak produktif jika selesai lebih cepat dari waktu Sprint 2. Ada *task* atau User Story yang tidak selesai dikerjakan.

				pengalaman melakukan Sprint Planning.	
3	Daily Scrum kurang efektif	Pelaksanaan proses Daily Scrum yang kurang sesuai dan mengakibatkan peserta Daily Scrum tidak mendapatkan manfaat dalam mengikuti Daily Scrum.	Manajemen Proyek	1. Durasi Daily Scrum yang terlalu lama. 2. Membahas masalah teknis didalam Daily Scrum. 3. Ketidakhadiran Product Owner didalam Daily Scrum.	1. Pengembang akan bosan dan tidak fokus. 2. *Stakeholder* yang tidak berkepentingan terhadap masalah teknis yang dibahas tidak akan mendapatkan *benefit.* 3. Memicu terjadinya *miscommunication.*
4	Sprint Retrospective tidak rutin diadakan	Tidak dilaksanakannya Sprint Retrospective yang merupakan salah satu event dalam kerangka kerja Scrum untuk membahas mengenai pelaksanaan Sprint sebelumnya.	Manajemen Proyek	1. Tidak adanya *dedicated* Scrum Master. 2. Sprint Retrospective dianggap memakan waktu.	1. Sulit menjalankan proses Scrum dengan baik. 2. Pengembang sulit mengeluarkan keluh kesah selama bekerja tanpa adanya Sprint Retrospective. 3. Tidak dapat belajar dari kesalahan.
5	Tambahan pekerjaan ditengah Sprint	Adanya pekerjaan tambahan berupa User Story baru	Manajemen Proyek	1. Adanya tambahan pekerjaan dari pihak manajemen.	1. Ada User Story yang harus mundur ke Sprint berikutnya.

		atau *task* baru yang dimasukan kedalam sebuah Sprint yan sedang berjalan.			2. Pengembang harus meninggalkan pekerjaannya untuk mengerjakan pekerjaan yang memiliki prioritas lebih tinggi.
6	Adanya *dependency*	Terjadinya kebergantungan atau saling tunggu-tungguan untuk dapat melanjutkan pekerjaan baik dalam skala antar tim maupun dalam skala antar *task*.	Organisasi	1. *Planning* yang kurang detail (prioritas *story* yang tidak memperhatikan kebergantungan *code*/teknis). 2. Kurangnya komunikasi/koordinasi antar anggota tim dan perbedaan prioritas pengerjaan *backlog* antar tim. 3. Pemecahan *task* yang besar menjadi lebih kecil.	1. Waktu menyelesaikan suatu fitur akan menjadi semakin lama.
7	Bekerja dengan tidak maksimal	Pengembang tidak dapat mengerjakan pekerjaan yang diberikan dengan maksimal karena beberapa alasan.	Organisasi	1. Scrum memiliki batas waktu pengerjaan pada setiap Sprint. 2. Pengembang mengerjakan pekerjaan dengan	1. Hasil pekerjaan desainer menjadi kurang baik. 2. Ada User Story/*task* yang tidak selesai. 3. Waktu merilis produk menjadi

				terburu-buru. 3. Pengembang sakit.	mundur.
8	Jumlah atau komposisi anggota tim tidak efektif	Keadaan tim Scrum baik dari jumlah anggota tim Scrum yang terlalu banyak atau komposisi tim yang kurang tepat sehingga membuat pekerjaan menjadi tidak efektif.	Organisasi	1. Jumlah anggota tim tidak optimal. 2. Komposisi tim yang tidak merata antara anggota junior dan anggota senior.	1. Terlalu banyak anggota tim menyebabkan banyak *communication channel* yang terbentuk. 2. Sulit mengatur banyak orang sekaligus. 3. Anggota junior sulit beradaptasi tanpa bimbingan senior.
9	Tidak adanya Produk Owner yang khusus	Tidak adanya seseorang dalam perusahaan yang menggunakan Scrum untuk menjabat sebagai produk owner yang khusus dan bekerja secara spesifik sebagai Product Owner.	Organisasi	1. Belum ada pengisi posisi Product Owner. 2. Jabatan Product Owner diberikan kepada pihak manajerial tertentu.	1. Kehabisan PBI karena tidak sempatnya membuat PBI. 2. Product Owner jarang hadir dan mengikuti proses Scrum karena jabatan manajerial nya menuntut untuk menghadiri rapat lain. 3. Turunnya moral pengembang.
10	Kurangnya keterlibatan QA	Quality Assurance yang merupakan	Organisasi	1. Kurangnya kesadaran	1. Memicu timbulnya *bugs.*

	didalam proses Scrum	salah satu pengembang tetapi tidak diikut sertakan dalam event Scrum baik itu di Sprint Planning, Daily Scrum, Sprint Review atau Sprint Retrospective.		tim*development* mengenai pentingnya keterlibatan QA dalam tim Scrum. 2. QA dianggap hanya diperlukan untuk melakukan *testingfeature.*	
11	*Meeting* tambahan yang mengganggu	Adanya *meeting* tambahan untuk pengembang diluar *meeting* yang merupakan acara Scrum namun dapat mengganggu pengembang dalam melakukan tugasnya untuk mengembangkan perangkat lunak.	Organisasi	1. Permintaan pihak manajemen. 2. Perubahan jadwal *meeting.* 3. Terlalu banyak *meeting.*	1. Mengganggu konsentrasi pengembang saat sedang bekerja. 2. Mengganggu jadwal kerja pengembang. 3. Mengganggu produktivitas pengembang. 4. Pekerjaan akan tertunda.
12	Kesalahan penafsiran terhadap *mindset* Agile	Adanya *stakeholder* yang tidak mengerti mengenai prinsip Agile dalam menggunakan Scrum sehingga penerapan Scrum	Organisasi	1. Menganggap Scrum sebagai mini Waterfall. 2. Kurangnya ketertarikan pengembang terhadap Scrum dan Agile.	1. *Stakeholder* menginginkan semua yang dikerjakan di Sprint dapat selesai semuanya sesuai waktu yang ditentukan. 2. Tidak ada rasa

		menjadi kurang efektif.		3. Pengembang hanya merasa sebagai eksekutor. 4. Pengembang belum memahami konsep Agile dan kerangka kerja Scrum.	kepemilikan terhadap produk yang dibuat.
13	Kebutuhan yang tidak jelas	Adanya kebutuhan yang dianggap kurang jelas bagi pengembang atau *stakeholder* terkait yang menghambat mereka dalam menghasilkan produk yang sesuai dengan ekspektasi.	Teknis	1. Pengembang tidak mengetahui kebutuhan proyek secara keseluruhan karena hanya diberikan kebutuhan per-Sprint. 2. Kebutuhan dibuat kedalam bentuk User Story yang kurang detail.	1. Pengembang belum bisa mendapatkan gambaran akhir mengenai produk yang akan dihasilkan. 2. Pekerjaan pengembang dapat menjadi tidak sesuai dengan ekspektasi Product Owner. 3. Acceptance Criteria yang kurang jelas memicu munculnya *bugs*.
14	Tujuan proyek tidak jelas	Tujuan dari proyek yang akan dibuat kurang dimengerti oleh pengembang sehingga pengembang sulit memperkirakan apa	Teknis	1. PO memberikan tujuan proyek yang terlalu umum. 2. PO kurang mendeskripsikan tujuan proyek.	1. Pengembang tidak mengerti tujuan dari proyek. 2. Hasil yang dibuat pengembang tidak sesuai dengan ekspektasi PO.

		yang harus dilakukan untuk menghasilkan produk yang sesuai dengan tujuan proyek.			
15	Kurangnya dokumentasi	Kurang atau bahkan tidak adanya dokumentasi mengenai produk yang dibuat baik dari pengembang maupun pihak manajemen sehingga menimbulkan beberapa masalah.	Teknis	1. *Stakeholder* terkait merasa malas untuk membuat dokumentasi. 2. Pengembang merasa dokumentasi merupakan pekerjaan yang melelahkan. 3. Dokumentasi dianggap membuang-buang waktu karena perubahan yang terjadi akan membuat dokumentasi harus berubah.	1. Proses *onboarding* karyawan baru menjadi lama. 2. Sulit mengingat kembali *feature* yang dibuat saat diperlukan oleh *stakeholder* tertentu. 3. QA sulit melakukan *regression test.*
16	Ketidakcocokan Pair Programming	Adanya rasa ketidakcocokan dalam melakukan Pair Programming dengan pengembang lain. Ketidakcocokan	Teknis	1. Pengaruh karakter *single fighter* dalam diri pengembang.	1. Proses *development* menjadi lambat.

		ini membuat pengembang tidak dapat bekerja sama dalam melakukan Pair Programming.			
17	Perubahan kebutuhan yang sangat cepat	Perubahan kebutuhan *market/user* yang sangat cepat sehingga perusahaan harus mengakomodir permintaan perubahan yang terjadi agar produk yang dibuat dapat tetap bersaing dipasaran.	Eksternal	1. Organisasi Agile memang adaptif terhadap perubahan. 2. Bekerja di perusahaan berbasis internet.	1. Task yang sudah masuk ke *done* dapat kembali lagi ke *work in progress.* 2. Tidak dapat melakukan *development* yang baik karena segala sesuatu dituntut cepat.

serviens in lumine veritatis