Midiendo la eficiencia del proceso Ágil

Un caso de estudio en GMV - EUSC (Centro de Control de Satélites Eutelsat)

Belén Moreno (As Scrum Master, GMV),
David López (As Product Owner, GMV),
Javier Garzás (As Agile Coach)

1. TL;DR

La mejora de la eficiencia del proceso de desarrollo debe ser parte integral de una cultura Ágil. En Scrum, el punto de historia se ha usado tradicionalmente para evaluar la velocidad y eficiencia intentando que, según pasan los sprints, el número de puntos historia completados (*working software*[1]) sea mayor. Sin embargo, aunque muchas veces sea necesaria e inevitable temporalmente, esta aproximación conlleva muchos desperdicios derivados.

En su lugar, en este artículo contamos nuestra experiencia utilizando medidas basadas en tiempos, como el *cycle time*, el *touch time,* el *waste time* y el *takt time*. Con estos tiempos definimos la eficiencia del proceso y proponemos estrategias para medirlo y mejorarlo. En este artículo presentamos un caso real dentro de un equipo Scrum y mostramos cómo nuestras medidas nos dan una visión realista de la eficiencia del proceso.

2. Contexto, quienes somos, de dónde venimos

GMV es un grupo empresarial tecnológico español, fundado en 1984, y que ofrece servicios y productos en sectores como la aeronáutica, el espacio y la defensa. GMV tuvo en 2018 1.846 empleados y ventas anuales de 196 millones de euros, siendo el 65% de estas ventas internacionales.

Desde 2011, GMV es el proveedor número uno de Centros de Control para satélites comerciales de telecomunicaciones, ya que la mayoría de los satélites lanzados en los últimos años están siendo controlados con el software GMV. Hay alrededor de 370 satélites que utilizan la tecnología de software terrestre GMV, operada por 32 operadores comerciales mundiales, incluidos SES (Luxemburgo), Eutelsat (Francia), Arabsat (Arabia Saudita), Hispasat (España), Türksat (Turquía), NBN Co (Australia) , Starone (Brasil) y muchos otros.

Este artículo es sólo una instantánea de la transformación ágil de EUSC (sección de GMV para el mantenimiento y desarrollo del Centro de Control de Satélites de Eutelsat, European Telecommunications Satellite Organization), en lo que refiere solo a cómo trabajamos ahora (seguramente, cuando lo leas, ya habremos evolucionado con nuevas ideas) en mejorar la eficiencia (aquí no entramos en temas de eficacia) del proceso Ágil.

Nuestra transformación ágil comenzó en octubre de 2016. Anteriormente, la unidad estaba estructurada en silos, grupos especializados, cada uno dirigido por un jefe de proyecto, el modelo de desarrollo seguía un ciclo de vida clásico en cascada con diagramas de Gantt y otros tantos clásicos de la gestión tradicional que hemos ido dejando atrás.

Además, nuestro sistema se basa en un software *legacy*, del que partíamos sin test automáticos (ni unitarios ni de integración), con más de 3 millones de líneas de código en C++ y Java.

3. Antecedentes: nuestra historia... con el punto historia

Cuando comenzamos usando Scrum, el punto historia nos ayudó enormemente. Con los puntos historia pudimos conocer cuántas historias podíamos abarcar en cada sprint y nos sirvieron para empezar a buscar mejoras en las retrospectivas, para incrementar el número de puntos que hacíamos. En definitiva, nos ayudaron a empezar con Scrum y a buscar la mejora continua desde el principio.

Sin embargo, con el tiempo fuimos conscientes de los desperdicios que implicaba el uso de los puntos historia. En particular, no dimos cuenta de que:

- Nuestra tendencia era crear *plannings* interminables. Llegaron a durar 4 y 5 horas por las largas discusiones que se desarrollaban, bajo técnicas como el *planning poker*, para asignar puntos a las historias.
- El hecho de estimar con más precisión se volvió un objetivo en sí mismo. Se empezó a dedicar esfuerzo a analizar la precisión de las estimaciones con punto historia, y en crear y mantener estadísticas para mejorarlas.
- Sólo llevar el número de puntos historia por sprint no era suficiente (incluso era peligroso), necesitábamos saber también el número de historias terminadas y cómo de precisos éramos a la hora de estimar (por ejemplo, usando curvas de Rayleight[2]) .

Todo esto nos llevó a la conclusión de que estimar las historias con puntos historia nos estaba costando un esfuerzo extra y conllevaba desperdicios.

Comenzamos entonces a trabajar en conseguir historias pequeñas y lo más parecidas en esfuerzo posible. Fue costoso, nos llevó tiempo, pero cuando lo conseguimos pudimos dejar de estimar las historias con puntos historia y empezamos a contabilizar el número de historias terminadas por sprint, lo que nos acortó los *plannings*, eliminó el *planning poker*, nos forzó a tener historias pequeñas, etc.

Al quedarnos sin el punto de historia, empezamos a trabajar con otras maneras de medir la eficiencia de nuestro proceso de desarrollo, para continuar así el proceso de mejora continua.

4. Cómo estamos midiendo ahora la eficiencia

Como ya comentamos, durante un tiempo sólo pudimos usar el punto historia para medir la eficiencia de nuestro proceso (básicamente porque las historias eran de tamaños muy diferentes). Pero ahora utilizamos principalmente las siguientes métricas:

- Cycle time (su traducción en castellano sería "tiempo de ciclo")
- Waste time ("tiempo de desperdicio")
- Touch time ("tiempo de trabajo")
- Takt time adaptado (podría ser algo como "ritmo necesario a alcanzar")

Estas métricas se basan en tiempos (medidos en horas o días) que las historias están en distintos estados, o en los tiempos que equipo de desarrollo les dedica en sus distintas etapas durante la duración del sprint.

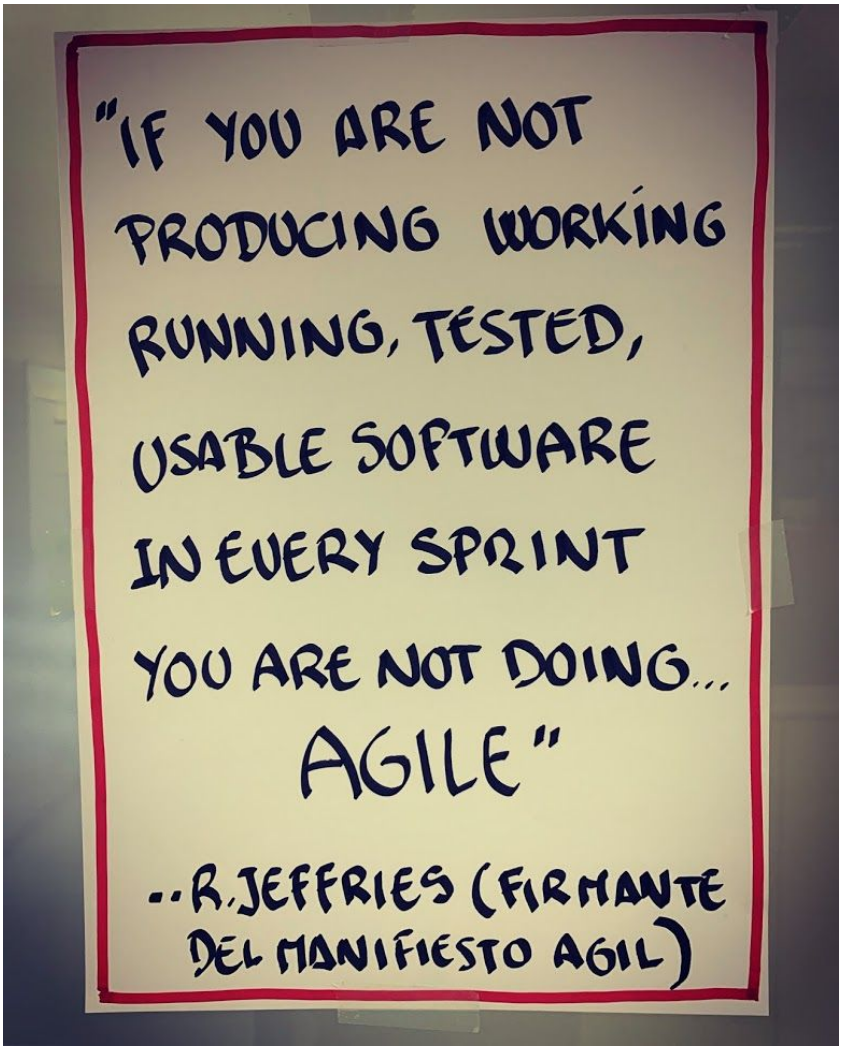
Nuestro objetivo es incrementar el tiempo que se dedica al desarrollo de las historias de forma eficaz, y buscar los desperdicios que nos "roban" tiempo y que hacen que no aportemos tanto valor al producto como podríamos en cada iteración.

Vamos a ir viendo cómo usamos cada una de ellas.

4.1. Cycle Time

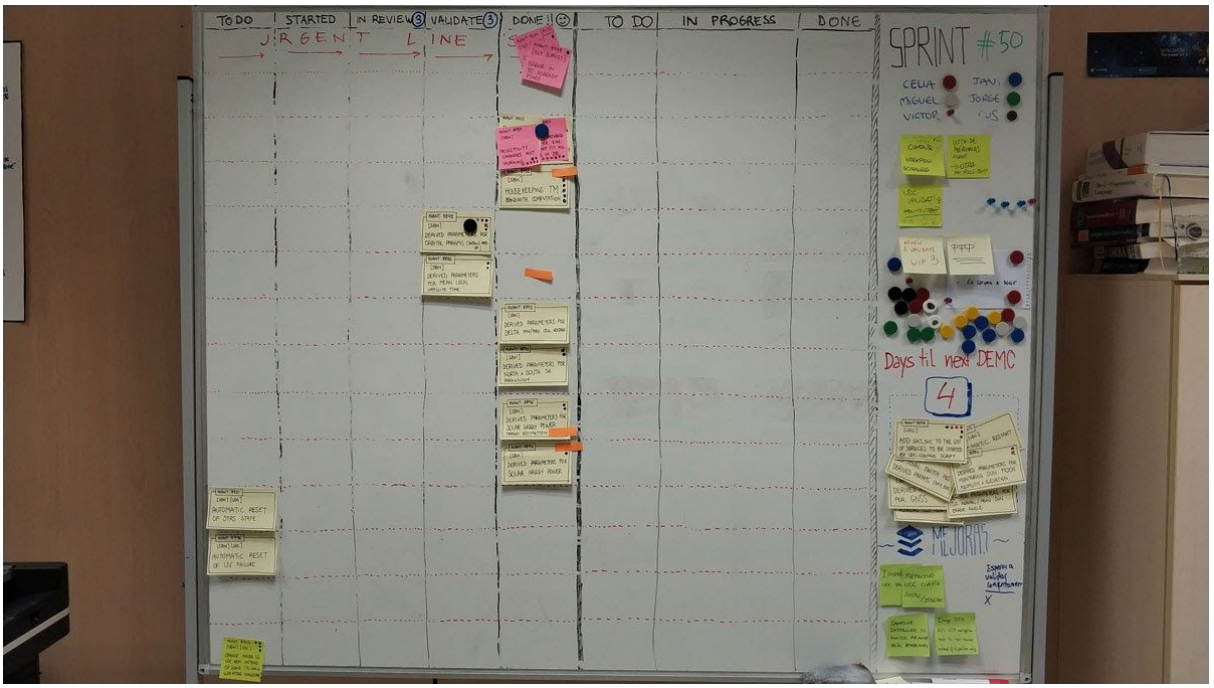
El *cycle time* es el tiempo que transcurre entre que se empieza a trabajar en una tarea hasta que se termina (básicamente desde el estado *doing* al estado *done*). Existen numerosas referencias[3] en Agilidad que recomiendan el uso de *cycle time*.

Para calcular el *cycle time*, los equipos deben realizar un seguimiento de cuándo comienzan a trabajar en la historia y cuándo está en terminada, en el *done* o, mejor aún… en entorno de producción o preproducción (working software).



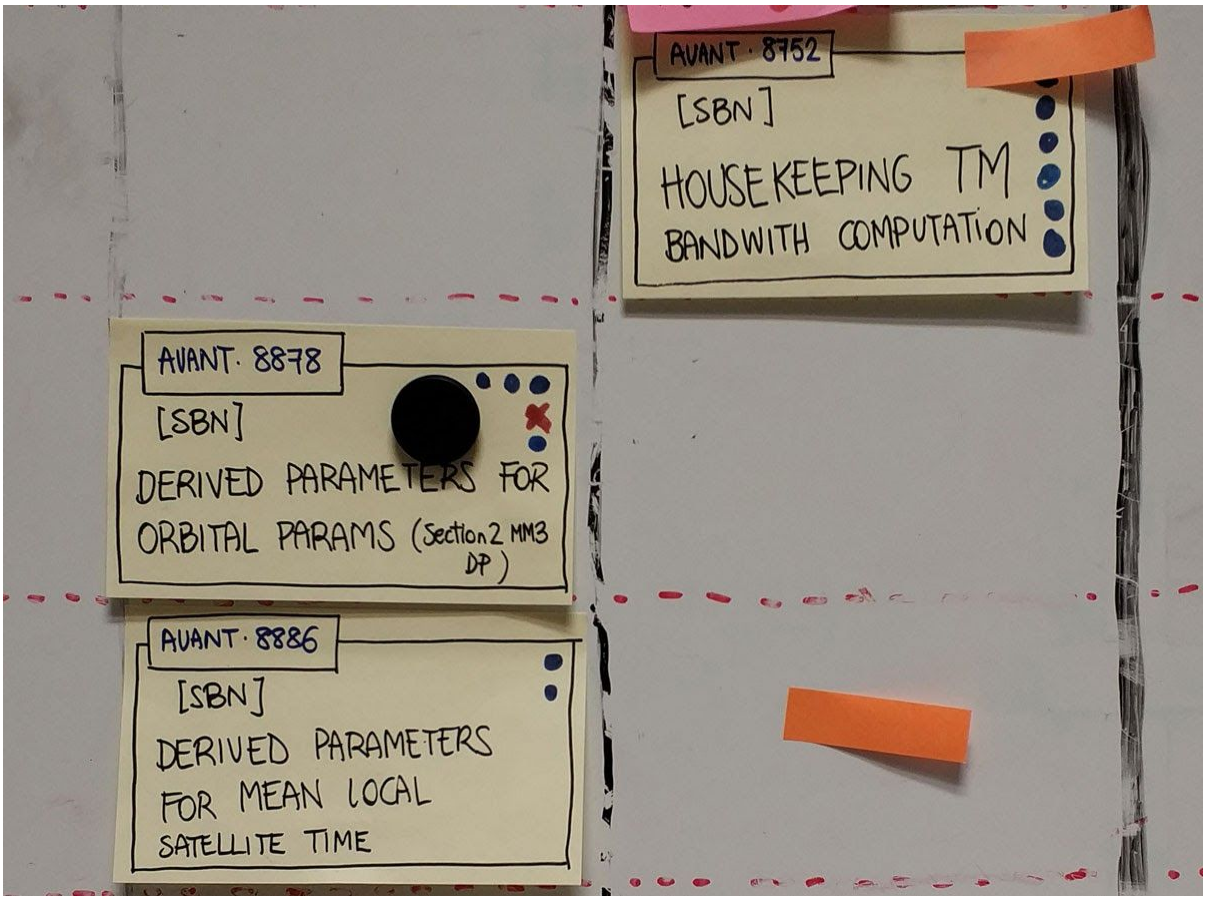
El *cycle time* representa la cantidad total de tiempo dedicado a realizar un *item* (principalmente nos interesan las historias de usuario, pero no nos limitamos solo a ellas, sino también medimos las *tasks*, *improvements* y *bugs*).

El *cycle time* se mide en días u horas (dependiendo del tamaño que suelan tener tus historias de usuario), y solo cuentan los días/horas laborales, no naturales. Y, ¡ojo!, el *cycle time* únicamente mide el tiempo transcurrido (no tiempo real trabajado sobre un *item*).



Tablero del equipo con *user stories* en amarillo

Lo que hacemos nosotros es en cada *daily* colocar un punto[4] en la tarjeta de cada *user story* si el día anterior se trabajó en ella. Y colocamos una aspa roja por cada día que no se trabajó estando “abierta” (se entiende como “abierta” cualquier estado entre *doing* y *done*).



Detalle de las X, que cuentan los días en los que no se trabajó en un *item*, y los puntos, que cuentan los días de trabajo

El término *cycle time* se utiliza normalmente como el valor promedio de todos los *cycle time* obtenidos en cada *item* del *sprint backlog*. Así, se puede hacer seguimiento de esta métrica a medida que completamos sprints.

Id	Type	Description	Cycle time (days)
AVANT-8801	Bug	[SBN] TC description not correctly shown in TC history	2
AVANT-8768	Story	[QTM] Processing of Reed-Solomon data in the Transfer Frame	1
AVANT-8691	Bug	[QTM] Cannot execute procedures in secondary satellite	0
AVANT-8486	Story	[SBN] Block application if system is muted onboard	10
AVANT-8807	Task	[QTM] Add redundant encryption unit to configuration	0
AVANT-7149	Story	[SBN] Onboard events additional information	9

AVANT-8672	Bug	[QTM][SBN] Parameters of 11 chars not fully shown in command edit window	2

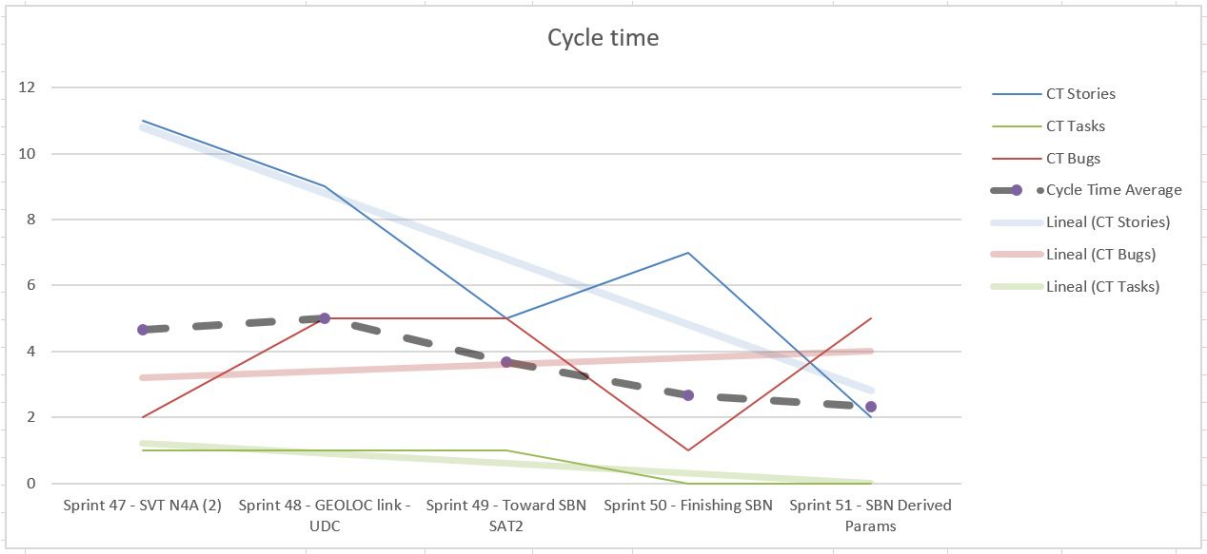
Ej. Cálculo del promedio de cycle time del equipo para el sprint #50. Cuando un cycle time nos da 0 significa que el equipo ha resuelto el issue en menos de un día de trabajo (ya que la unidad que estamos utilizando para medir es "días", aunque se podría utilizar "horas" también)

Aquí no solo tenemos cifras de las *user stories* sino de *tasks* y *bugs* también, porque es lo que el equipo completó, para este ejemplo, en el sprint #50. Si agrupamos por tipo de *issue*, podemos ver los promedios. En el sprint #50 (2 semanas = 10 días de trabajo), el equipo tardó de media 7 días en resolver cada *user story*, de media 1 día en resolver cada *bug* y tardó de media menos de 1 día en resolver cada *task*:

Story		Bug		Task	
Id	**Cycle time (days)**	**Id**	**Cycle time (days)**	**Id**	**Cycle time (days)**
AVANT-8786	1	AVANT-8801	2	AVANT-8807	0
AVANT-8486	10	AVANT-8691	0		**0**
AVANT-7149	9	AVANT-8672	2		
	7		**1**		

Si queremos tener una métrica histórica de los últimos sprints, podemos calcular los *cycle time* por cada uno para agruparlos todos después y ver la progresión:

		Cycle time (days)			
Sprint Id	**Name**	**Story**	**Tasks**	**Bugs**	**Average**
246	Sprint 47 - SVT SAT1	11	1	2	5
249	Sprint 48 - GEOLOC link	9	1	5	5
252	Sprint 49 - Towards SBN SAT2	5	1	5	4
255	Sprint 50 - Finishing SBN	7	0	1	3
257	Sprint 51 - SBN Derived Params	2	0	5	2
		7	**1**	**4**	**4**



La tendencia general (Cycle Time Average - línea discontinua negra) en los 5 últimos sprints es favorable.

4.2. Waste Time y Touch Time

El *waste time* (o tiempo de desperdicio) representa la suma de todas las veces que el equipo fue interrumpido, o se quedó esperando a algún *input* necesario, bloqueos o cualquier otra pérdida de tiempo mientras agregaba valor a una historia.

El *waste time* también se mide en días u horas, y solo cuentan los días u horas de trabajo, similar al *cycle time*.

Podríamos medir el tiempo que el equipo dedica a trabajar sobre una historia (en vez del tiempo en el que no se trabaja sobre ella), pero observamos que medir esto a la gente le gusta menos que medir cuándo no se les deja trabajar.

Actividades que añaden valor directamente al *item* son tiempo real de trabajo que nos interesa maximizar, porque el resto… es desperdicio. Por ejemplo, los cambios de contexto y los tiempos de espera para revisar código o validar funcionalidad suelen llevarse la mayoría de este tiempo[5]. Por suerte, simplemente haciendo visible esta realidad ya hace que podamos tomar acciones sobre estos situaciones que nos hacen ser menos eficientes.

Sprint Id	Id	Type	Cycle time (days)	Touch time (days)	Waste time (days)
246	AVANT-58	Story	1	1	0
246	AVANT-8129	Story	34	29	5
246	AVANT-8182	Story	6	4	2
246	AVANT-8184	Story	6	0	6
246	AVANT-8188	Bug	5	2	3
246	AVANT-8317	Story	4	4	0
246	AVANT-8336	Bug	6	1	5
246	AVANT-8394	Bug	6	1	5
246	AVANT-8426	Bug	0	0	0
246	AVANT-8451	Task	0	0	0
249	AVANT-7567	Story	7	0	7

Descubrimos el *touch time* en el libro “Lean From The Trenches”[6] y nos pareció muy interesante incluir también esta métrica.

Si el *cycle time* se calcula como cuántos días tardó un *item* en cruzar el tablero del proyecto, el *touch time* mide cuántos de esos días fueron de trabajo real (cuántos días se "tocó" el *item*).

En la tabla anterior podemos ver la diferencia entre el *cycle time* real de cada *item*, y el tiempo trabajado por el equipo (*touch time*). Por tanto, si restamos una a la otra, podemos ver el *waste time* en cada una.

Si ahora ordenamos de mayor a menor la columna de *waste days*, podemos fijarnos en qué *issues* son los que estuvieron más tiempo parados y por tanto, donde hubo más desperdicio.

Sprint Id	Id	Type	Cycle time (days)	Touch time (days)	Waste time (days)
255	AVANT-8486	Story	16	2	14
249	AVANT-8407	Bug	14	2	12
249	AVANT-8131	Story	28	16	12
257	AVANT-8757	Bug	14	2	12
252	AVANT-8148	Bug	17	6	11
249	AVANT-8356	Story	10	0	10
249	AVANT-8438	Bug	9	2	7
249	AVANT-7567	Story	7	0	7
249	AVANT-8166	Bug	9	2	7
257	AVANT-8655	Bug	9	2	7
252	AVANT-8484	Story	10	4	6
246	AVANT-8184	Story	6	0	6
246	AVANT-8336	Bug	6	1	5
257	AVANT-8752	Story	7	2	5
257	AVANT-8882	Story	5	0	5
246	AVANT-8129	Story	34	29	5
246	AVANT-8394	Bug	6	1	5
249	AVANT-8474	Story	5	1	4

Issues ordenados descendentemente por *waste time*

Es curioso ver que hay casos, como la *user story* AVANT-8129 de 34 días de *cycle time* (la tercera por abajo), a pesar de tener el mayor *cycle time* de todas, no es la que tuvo el mayor desperdicio, ya que el *touch time* aquí fue de 29 días.

4.3. La eficiencia del ciclo y del proceso

Los anteriores datos son muy interesantes; nos sirven para reflexionar sobre cómo hay *items* que tardaron 10 días en pasar del *doing* al *done* del tablero, pero realmente sólo se trabajó en ellos 3 días. En este ejemplo, 7 días fueron de tiempos de esperas, ¡una locura si queremos ser eficientes!

Y con los anteriores calculamos la eficiencia del proceso o nivel de ciclo. Según Jeff Sutherland[7] la eficiencia del proceso es el tiempo dedicado a agregar valor a la historia, como un porcentaje del tiempo total dedicado a la historia. Y lo calculamos así:

$$Cycle\ ó\ Process\ efficiency\ = \frac{Cycle\ Time - Waste\ Time}{Cycle\ Time}$$

Con nuestro ejemplo anterior, tendríamos un 30% de eficiencia en el proceso. Intentar aumentar este número es una excelente manera de descubrir y eliminar el desperdicio.

		Days			
Id	**Name**	**Cycle time**	**Touch time**	**Waste time**	**Cycle efficiency**
AVANT-8801	Bug	2	1	1	50%
AVANT-8768	Story	1	1	0	100%
AVANT-8691	Bug	0	0	0	100%
AVANT-8486	**Story**	**10**	**2**	**8**	**20%**
AVANT-8807	Task	0	0	0	100%
AVANT-7149	Story	9	7	2	78%
AVANT-8672	Bug	2	1	1	50%

Datos de los *issues* terminados en el sprint #50

Si sacamos el promedio de la eficiencia del ciclo de todas las *user stories* de cada sprint, podríamos ver la evolución a lo largo del tiempo de nuestra reducción de desperdicio.

4.4. Takt Time adaptado

El término "Takt" proviene del alemán y significa "pulso" en castellano. En los mundos de *Lean Manufacturing*[8,9] se utiliza esta métrica para saber cuánto tiempo debes tardar en producir un solo *item* de tu producción para ser competitivo y tener contento a tu cliente.

Supongamos que tenemos una fábrica de ordenadores, y que para fabricar un sólo ordenador por completo tardamos actualmente 30 minutos. Si trabajamos 8h/día, cada día podemos fabricar 16 ordenadores. Al cabo de un mes, ya que trabajamos 20 días porque no contamos los fines de semana, fabricamos 320 ordenadores.

Resulta que nuestro cliente necesita que le vendamos 500 ordenadores al mes o se va a la competencia, y por razones obvias necesitamos llegar a esas cifras. Por tanto, 500 ordenadores entre 20 días nos dice que al día tenemos que fabricar 25 ordenadores. Como no queremos hacer horas extra y lo que queremos es ser más eficientes, debemos llegar a producir un ordenador en 19 minutos.

Actualmente, tardamos 30 minutos en completar un item. ***Cycle Time = 30 min***
Idealmente, debemos tardar 19 minutos en completar un item. ***Takt Time = 19 min***

Debemos reducir nuestro tiempo de producción en 11 minutos (37% del tiempo que ahora tardamos) por item analizando interrupciones y bloqueos tiene nuestro sistema de producción actualmente y eliminarlos por completo.

Como las *user stories* de un equipo auto-organizado no se completan igual que unidades en una fábrica (recuerda, hacemos software y los ejemplos de la fabricación hay que tomarlos con cuidado, muchas cosas no aplican), hay que adaptar este término *takt time* a nuestro marco de trabajo para sacarle valor.

Podríamos pensar que si actualmente nuestra velocidad es de 10 historias en sprints de 2 semanas de duración (10 días hábiles dejando fines de semana fuera), nuestro *takt time* sería de 1 día para acabar una historia de usuario si nuestro WIP = 1 (en un día deberíamos completar 1 historia para tener un *sustainable pace* como nos aconseja XP[10]).

Como nuestro equipo tiene 5 desarrolladores no tiene sentido tener un WIP tan pequeño porque no todas las historias se pueden partir para hacer tanto *swarming*[11]; así que supongamos que tenemos un WIP = 3. Por tanto, podríamos decir que:

$$0,5 <= \ takt\ time \ <= \ 0,5 * 3$$

Así podríamos tener un dato orientativo de cuánto nos podemos permitir tardar en hacer una sola historia (máximo *cycle time* permitido por nosotros mismos). Sería sano si tardáramos entre medio día (0,5) y día y medio (1,5) en completar cada *user story*. El *takt time* lo usamos, por tanto, como dato de referencia para mantenernos en el buen camino a medida que avanza el sprint.

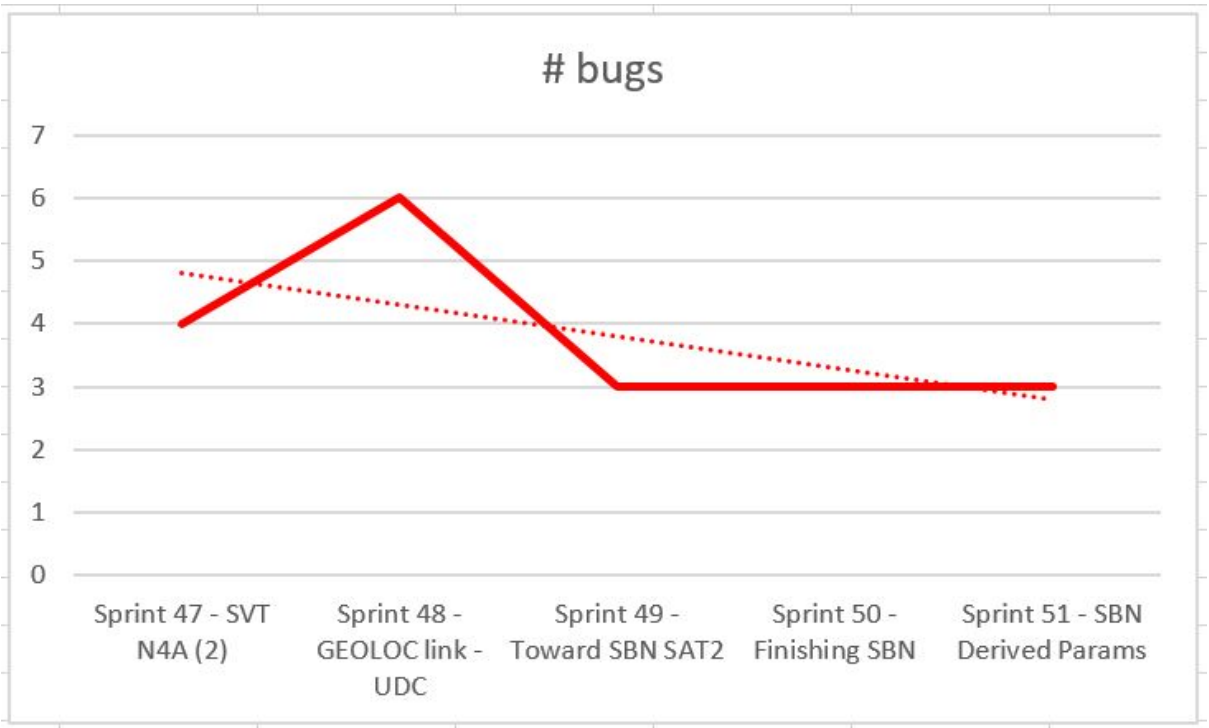
Y también para concienciarnos de ir a historias de usuario pequeñas, que entren en el rango que buscamos.

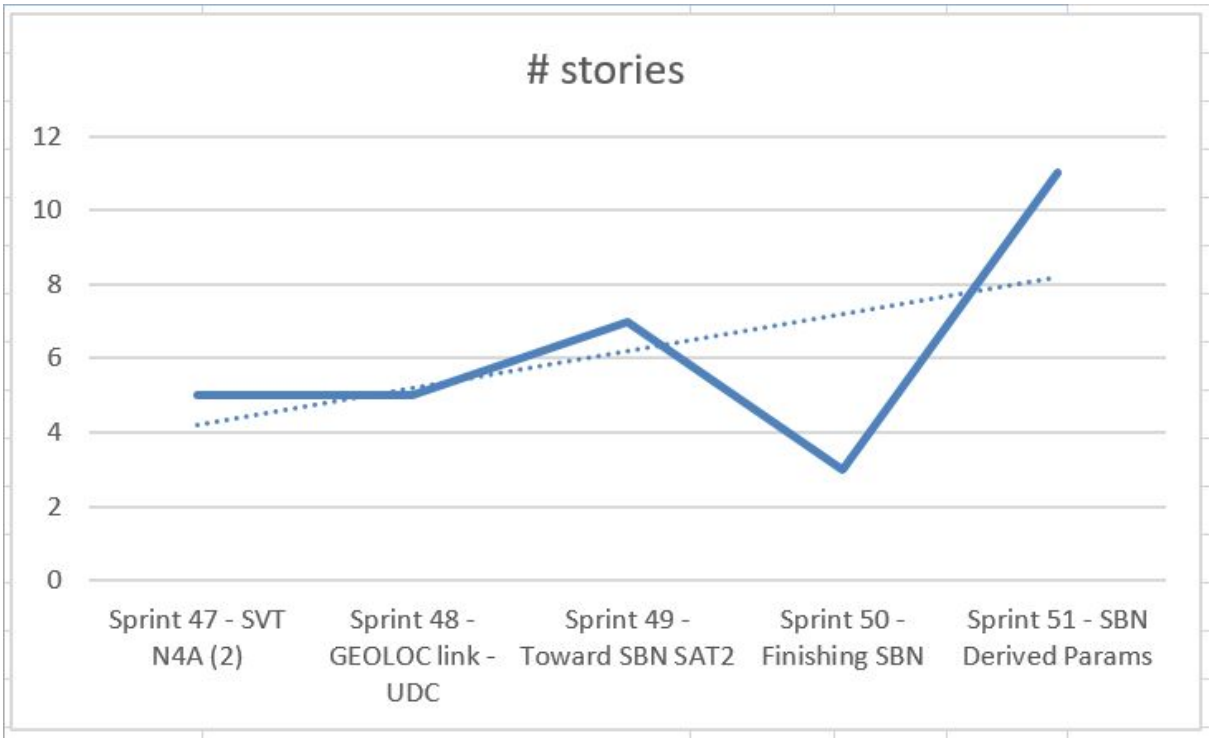
5. Mejorando la eficiencia del proceso Ágil

Nuestro objetivo es tardar menos en hacer "el mismo trabajo" y hacerlo bien (control de deuda técnica) para ser eficientes, ¿y para qué? pues no hay que perder el objetivo... para hacer más *user stories* si es posible.

Por ello, además de querer minimizar el *waste time* para reducir el *cycle time*; analizar si podemos reducir el *touch time* aún más; tener en mente el *takt time* para que nos guíe a alcanzar este objetivo… También es interesante medir si en cada sprint hacemos más *user stories* y menos *bugs*.

Sprint Id	Sprint name	# stories	#bugs	#tasks
246	Sprint 47 - SVT SAT1	5	4	1
249	Sprint 48 - GEOLOC link	5	6	0
252	Sprint 49 - Toward SBN SAT2	7	3	0
255	Sprint 50 - Finishing SBN	3	3	1
257	Sprint 51 - SBN Derived Params	11	3	1


bugs
7
6
5
4
3
2
1
0
Sprint 47 - SVT N4A (2)
Sprint 48 - GEOLOC link - UDC
Sprint 49 - Toward SBN SAT2
Sprint 50 - Finishing SBN
Sprint 51 - SBN Derived Params


stories
12
10
8
6
4
2
0
Sprint 47 - SVT N4A (2)
Sprint 48 - GEOLOC link - UDC
Sprint 49 - Toward SBN SAT2
Sprint 50 - Finishing SBN
Sprint 51 - SBN Derived Params

Hasta ahora hemos estado hablando de la eficiencia del proceso sin profundizar en qué cosas son realmente las que hacen que seamos más eficientes. Niklas Modig, en el libro de “This is Lean”[12] presentaba dos dimensiones en las que podemos orientar la mejora:

- Eficiencia del equipo (minimizar interrupciones, tomas de decisiones más rápidas, etc.)
- Eficiencia del flujo (automatizar tests, mejorar nuestro sistema de Integración Contínua, etc.)

Básicamente, lo que nos dice esta matriz es que no basta con mejorar solo una de las dos dimensiones, sino que hace falta poner el foco en las dos e ir a la par para conseguir ser más eficientes realmente.

Analizar el *waste time* para reducirlo al mínimo sería una de las direcciones (eficiencia del equipo); pero analizar el *touch time* y ver cómo podemos llegar a alcanzar el *takt time* sería otra (eficiencia del flujo).

Sin querer extender este artículo, y como conclusión, queríamos compartir contigo algunos ejemplos de acciones que hemos experimentado (cada equipo es un mundo, lo que a nosotros nos funciona puede que ati no, esto va de experimentar, ¿no?) en nuestro equipo para poder trabajar en ambas dimensiones de mejora:

Acción de mejora	Eficiencia del equipo	Eficiencia del flujo
Revisiones de código y validaciones cruzadas		*
Time-boxear cada reunión formal e informal, y tener un moderador que haga que se cumpla	*	
Establecer un WIP y respetarlo		*
Utilizar semáforos para limitar interrupciones	*	
Pair programming en user stories complejas -a criterio del equipo- y la pareja se encarga de llevar la historia hasta el *done*: se implementa, se revisan y se valida todo en pareja		*
Utilizar un token al hablar para respetar turnos de palabra	*	
Configurar en *Bamboo* el poder lanzar despliegues automáticos desde ramas y que corran los tests automáticamente		*
Definition of Ready más estricto	*	
Refactorizar el código de build para que tarde un 30% menos de lo que tarda		*
Hacer sesiones de formación dentro del equipo, unos a otros	*	
Esforzarnos por partir más las *user stories* y entonces hacer más en cada sprint		*
Motivar al equipo con diferentes actividades o hacer cosas nuevas	*	
Si hay mucha variabilidad de velocidad de sprint a sprint, puede ser por mucha deuda técnica acumulada. Dedicar dentro de cada user story un poco más de tiempo a refactorizar y dejar el código siempre “mejor de lo que me lo he encontrado”		*
Dejar de estimar y aprovechar ese tiempo en sacar más historias de usuario	*	

Agradecimientos

Con gran aprecio y respeto, a los autores de este artículo nos gustaría agradecer el esfuerzo y la ilusión que están poniendo todas las personas de EUSC (Centro de Control de Satélites de Eutelsat) en este camino de cambio (con mención especial al equipo protagonista de este artículo, por supuesto, con cariño para vosotros), y por cómo han contribuido directa o indirectamente a esta transformación Ágil.

De izquierda a derecha, Javier Garzás, David López y Belén Moreno

6. Referencias

[1] Wells, D. (2009). *Working software*. Agile process: Agile-Process.org. http://www.agile-process.org/working.html

[2] Garzás, J. (2018). *Rayleigh para Dummies (y qué tiene que ver con la estimación ágil) (1/2)*. Javier Garzás: javiergarzas.com. From https://www.javiergarzas.com/2018/03/la-curva-de-rayleigh-para-dummies-1.html

[3] Skarin, M. (2009). *Why cycle time can tell you more than velocity*. Crisp: Crisp's Blog. From https://blog.crisp.se/2009/09/06/mattiasskarin/1252262279228

[4] Janlén, J. (2018). *Toolbox for the Agile Coach - Visualization Examples*. Leanpub: https://leanpub.com/agiletoolbox-visualizationexamples.

[5] Poppendieck, M., Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit (The Agile Software Development Series)*. United States: Addison Wesley.

[6] Kniberg, H. (2011). *Lean from the Trenches: Managing Large-Scale Projects with Kanban*. United States: The Pragmatic Programmers, LLC.

[7] Verbruggen, F., Sutherland, J., van der Werf, J. M., Brinkkemper, S., & Sutherland, A. (2019). 52nd Hawaii International Conference on System Sciences: *Process Efficiency - Adapting Flow to the Agile Improvement Effort*. From https://doi.org/10.24251/hicss.2019.837

[8] Wikipedia: Lean Manufacturing takt time. https://es.wikipedia.org/wiki/Takt_time

[9] Garzás, J. (2020). *Lead y Cycle time, midiendo la eficiencia del proceso Ágil*. Javier Garzás: https://www.youtube.com/watch?v=lqhhD3dgRrQ

[10] Jeffries, J. (2011). *What is Extreme Programming?*. Ron Jeffries: XProgramming. From https://ronjeffries.com/xprog/what-is-extreme-programming/

[11] Garzás, J. (2016). *El importante y desconocido Swarming, "flujo continuo de una sola pieza" o One-Piece Continuous Flow*. Javier Garzás: javiergarzas.com. From https://www.javiergarzas.com/2016/11/swarming-one-piece-continuous-flow.html

[12] Modig, N., Ahlstrom, P. (2012). *This is Lean: Resolving the Efficiency Paradox*. Sweden: Rheologica Publishing