RESUMEN DEL LIBRO BASADO EN LA METODOLOGÍA DE BASECAMP PARA DESARROLLAR PRODUCTOS DIGITALES Y PROYECTOS DE SOFTWARE

SHAPE UP: STOP RUNNING IN CIRCLES AND SHIP WORK THAT MATTERS

AUTOR DEL LIBRO: RYAN SINGER

AUTOR DEL RESUMEN:

Danny Prol: Twitter | Sitio Web

PRÓLOGO DE JASON FRIED

Twitter de Jason

- La manera en la que el equipo trabaja tiene una enorme influencia en lo que puede llegar a hacer:
    - Los procesos
    - Las metodologías
    - Las buenas prácticas
    - El enfoque
    - La disciplina
    - La confianza
    - El estilo de la comunicación
    - El ritmo (*pace* en inglés)

- La ejecución no lo es todo y menos en la industria del software. Ejecutar algo de mala manera puede destruir la moral y erosionar la confianza en el largo plazo. Vale, lo has ejecutado pero, ¿a qué coste?

- La gente normalmente se pregunta cómo Basecamp produce tan buen software en tan poco tiempo y logra mantener una retención de empleados tan alta

por años. No usan metodologías. No llenan las paredes de post-its. No hacen daily standups, sprints, o nada que se le parezca. Nada de backlogs, Kanban, etc.

- El enfoque de Basecamp: a lo largo de 15 años han ido desarrollando, a través de prueba y error, un proceso propio. Este libro es precisamente lo que quiere contar: el proceso documentado para los que quieran hacer las cosas de manera diferente. Este libro es para exploradores y pioneros que no siguen modas. Aquellos que quieren trabajar *mejor* que el resto.

AGRADECIMIENTOS

- Los diseñadores y programadores de Basecamp han sometido a prueba y mejorado las técnicas de Shape Up durante años para entregar proyectos de software reales. Sus esfuerzos hacen de este un libro práctico, no teórico.

Introducción

1. Este libro es una guía de cómo hacen desarrollo de producto en Basecamp. Es también una *caja de herramientas* llena de técnicas que puedes aplicar a tu manera.

Dolores de crecer: A medida que el equipo crece, algunos problemas aparecen:

- Los miembros del equipo sienten que los proyectos no tienen fecha de inicio y fin. Crecen sin término.
- PMs no encuentran tiempo para pensar estratégicamente sobre el producto.
- Los fundadores se preguntan: “Por qué no podemos sacar funciones como lo hacíamos cuando éramos pocos?”

Principales ideas recogidas en el libro:

- **Ciclos de seis semanas:** suficientemente largo para crear algo significativo de principio a fin y suficientemente corto para que cualquiera pueda sentir el deadline desde el comienzo. La mayoría de las nuevas funciones de Basecamp están creadas y lanzadas en ciclos de seis

semanas. Las decisiones de Basecamp están enfocadas en mover el producto hacia adelante. No les importan las horas o cómo el empleado invierte sus tiempos. No tienen reuniones diarias. No repiensan el roadmap cada dos semanas. Su foco está a un nivel más alto: “Si el proyecto se lanza en seis semanas, estaremos felices. Sentiremos que hemos invertido bien el tiempo”.

- **Shaping the work:** un equipo pequeño formado por gente Senior trabaja en paralelo con los “equipos de los ciclos”. Este equipo experimentado define los elementos clave de una solución antes de que en Basecamp consideren un proyecto “preparado”. Los proyectos están definidos de manera concreta para que los equipos sepan qué hacer en específico pero con el correcto nivel de abstracción para que tengan una “habitación” para trabajar en detalles que consideren interesantes. Cuando hacen “shaping”, ponen el foco menos en las estimaciones y más en su **apetito**. En vez de preguntar cuánto tiempo tomará sacar algo preguntan cuánto tiempo están dispuestos a invertir o cuánto vale la pena la idea. La tarea de “shaping” es: concretizar el problema y diseñar la solución de acuerdo a lo se ajuste a su apetito.

- **Empoderar a los equipos:** En Basecamp dan responsabilidad total a los equipos pequeños integrados por diseñadores y programadores. Ellos definen sus propias tareas, hacen ajustes al alcance o scope y trabajan juntos. No hay managers controlando o persiguiendo al programador. Los equipos se vuelven así más autónomos, y los Senior tienen que invertir menos tiempo en management poniendo el foco en “shape up” mejores proyectos. Cuando los proyectos están mejor “shaped”, los equipos tienen límites más claros y esto lleva a más autonomía.

- **Midiendo los riesgos:** En cada etapa del proceso miden el riesgo específico: el riesgo de no lanzar a tiempo. Ryan deja claro que mejorar el proceso debería ser posterior a tener la habilidad para lanzar. No sirve tener la mejor estrategia si no sabes cómo actuar en ella. Lo resuelven haciendo las preguntas correctas y resolviéndolas a tiempo. No dan proyectos a los equipos que estén atados a interdependencias.

Cómo está organizado el libro

1. **Shaping:** el trabajo previo que hacen en Basecamp con los proyectos antes de considerarlos preparados para agendar. Cada capítulo se para en un aspecto del proceso, desde configurar el apetito de una idea en bruto, pasando por pasar a Sketch una solución hasta escribir el pitch que represente el proyecto potencial. Explican las diferentes técnicas para mantener el diseño al nivel correcto de abstracción.
2. **Betting:** eligen entre los proyectos presentados y deciden qué van a hacer durante seis semanas.
3. **Building:** se refiere a las expectativas que ponen en los equipos y las prácticas especiales que usan para descubrir qué tienen que hacer. En este punto, ponen el punto de mira en cómo los equipos en específico descubren qué hacer, cómo integran las disciplinas de diseño y programación, cómo miden los desconocidos y los no conocidos, y finalmente cómo ellos administran el tiempo para acabar los proyectos a tiempo.

En el apéndice hay algunos consejos interesantes para poner en práctica los ciclos de seis semanas, cómo ajustar las técnicas según el tamaño de tu compañía, e indicaciones específicas de cómo implementar “Shape Up” usando Basecamp.

Parte 1 - Shaping

1. Nivel correcto de abstracción: ni muy vago ni muy concreto. Los wireframes son muy concretos y es definir a nivel detalle demasiado temprano. Esto no deja espacio para la creatividad.
2. Especificar demasiado el diseño lleva a errores de estimación. Aunque suene contraintuitivo, cuanto más específico, más difícil es de estimar. Cuando el scope no es variable, el equipo no puede reconsiderar una decisión de diseño que cuesta más de lo que vale.
3. Palabras demasiado abstractas: cuando el proyecto se define en pocas palabras, nadie sabe qué significa. “Build a calendar view” es abstracto. Los miembros del equipo no pueden decidir sobre esto. Resolver un problema sin contexto es un rollo. Tiene que quedar claro cómo tiene que funcionar y qué necesita ser construido exáctamente.
4. La labor de “shape” tiene 3 propiedades:
    a. Los programadores y diseñadores necesitan espacio para aplicar su propio juicio y experiencia

    b. Todos los elementos de la solución se pueden ver a un nivel macro y se pueden conectar. Hay una dirección clara hacia dónde se va.
    c. Indica qué NO hacer. Completar el proyecto en un tiempo determinado requiere limitar el alcance. Así reducen el riesgo.
5. La labor de shaping requiere combinar ideas en sketch o diagramas, con posibilidades técnicas y con prioridades de negocio. Shaping es trabajo de diseño en primera instancia. Define qué hace la funcionalidad, cómo funciona y cómo se encaja en los flujos existentes. No necesitas programar pero sí tener sensibilidad hacia lo técnico porque necesitas juzgar lo que es posible y lo que no. Necesitas ser estratégico también. Qué estás tratando de resolver? Por qué importa? Qué significa éxito? A qué clientes afecta? Cuál es el coste?
6. Dos caminos: uno para shape y otro para construir. Durante el ciclo de seis semanas, los equipos crean algo que ha sido previamente “shapeado” y los que “shapean” están trabajando en lo que podrían potencialmente trabajar en un futuro los otros. El trabajo de los que hacen shape no se comparte hasta que esté listo para mandarlo al otro equipo. Esto da la opción de poner el trabajo de shapers como WIP.
7. Los **pasos de shaping** son 4:
    a. **Fijar los límites:** cuánto vale la pena la idea, y cómo definimos el problema.
    b. **Dibujar la solución:** Alto nivel de abstracción. Apetito pero sin entrar demasiado en detalle.
    c. **Apuntar los riesgos:** cuando creemos que tenemos la solución, miramos los problemas que pueden emerger. Tarea de prevención de riesgos para evitar que el equipo creador pierda el tiempo.
    d. **Escribir el pitch:** formal. Resume el problema, limitaciones, solución y riesgos. Si el proyecto se elige, el pitch puede reusarse para explicar el proyecto al equipo.

Fijar los límites

1. Hablamos de una mejora menor o de un rediseño mayor? Siempre empezamos por una idea en bruto de los usuarios para definir la funcionalidad. Antes de entrar a discutir las maneras de resolver el problema, establecemos algunos límites para hacer la discusión más productiva.

2. Calma la excitación de crear algo nuevo viendo si realmente es algo que valga la pena invertir tiempo. Definir explícitamente cuánto tiempo del ciclo merece esa funcionalidad.
3. El **apetito** se define como un presupuesto de tiempo para un equipo de tamaño standard. Dos tamaños:
    a. Pequeño batch: 1 diseñador y 2 programadores
    b. Gran batch: el mismo equipo pero seis semanas completas
4. Tiempo fijo, alcance variable: el apetito no se parece en nada a la estimación. Estimaciones empiezan con una fecha de principio y final determinada. El apetito empieza con un número y termina con un diseño. Usan el apetito como una limitación creativa en el proceso de diseño.
5. El apetito limita el tiempo de soluciones que diseñamos durante el proceso de shaping. Cuando el equipo se pone manos en la masa, el tiempo fijo les empuja a tomar decisiones sobre lo que es core y lo que es innecesario.
6. Lo bueno es relativo en este caso: no hay definición absoluta de la mejor solución. Es relativo a los límites. La cantidad de tiempo que configuremos para el apetito nos llevará a diferentes soluciones. Solo podremos juzgar lo que es una buena solución en el contexto de cuánto tiempo queremos invertir y cómo de importante es la funcionalidad.
7. La respuesta por defecto a las ideas: Interesante. Igual algún día. Un no soft que mantenga las opciones abiertas. No lo ponemos en el backlog aún. Así damos espacio para aprender si es importante. No podemos comprometernos si todavía no entendemos cómo abordarlo. Es demasiado temprano para decir sí o no, primero shaping. Tampoco podemos tumbarla. No podemos mostrar mucha excitación y entusiasmo porque puede dar la sensación de que va a pasar.
8. Además de establecer el apetito, necesitamos aterrizar nuestro entendimiento del problema. Si atendemos a los desconocidos nos salen problemas que drásticamente afectan al alcance.
9. Puede que no llegues a la solución de primeras, pero tienes que sentir que el problema se vuelve suficientemente específico como para encajar en el apetito. El apetito te puede decir cuánta investigación hace falta.
10. Los rediseños o refactorizaciones que no están conducidas por un problema único o caso de uso no son proyectos. Es muy complejo determinar qué significa, cuándo inicia, cuándo termina. Pregunta qué no

está funcionando? En qué contexto? Qué partes del diseño existente puede permanecer idéntico y qué partes necesitan un cambio?

11. Se trata de saber qué significa que está en “done”. Configurar el apetito y tener claras expectativas en cada proyecto.
12. Cuando tenemos una idea en bruto, apetito y un problema definido estamos listos para movernos al siguiente paso y definir los elementos de la solución.

Encontrar los elementos. Dibujar la solución.

1. Es momento de pasar de las palabras o narrativa a los elementos de una solución de software.
2. Personas correctas con el mismo conocimiento y con quien puedas ser franco con ideas.
3. Evitar bajar al detalle en los dibujos y sketches. No nos importan los detalles innecesarios. El reto aquí es ser lo suficientemente concreto como para progresar en una solución específica. Las preguntas que tratamos de contestar son:
    a. Dónde encaja esta nueva funcionalidad en el sistema actual?
    b. Cómo lo hacemos?
    c. Cuáles son los componentes clave y las interacciones?
    d. Dónde nos lleva como usuarios?
4. Se usan técnicas de prototipado: “breadboarding” y “fat marker sketches”. Esto nos permite dibujar rápidamente diferentes versiones de flujos completos y debatir los pros y contras de cada enfoque y permanecer alineados.
5. Breadboarding: usamos un concepto de la ingeniería eléctrica que nos ayuda a diseñar con el nivel correcto de abstracción. La idea es plasmar los componentes clave y conexiones de la interfaz sin especificar el diseño visual en particular. Se basan en 3 dibujos:
    a. Lugares: navegación -> pantallas, diálogos, pop-ups...
    b. Affordances: lo que se puede accionar -> botones, campos. Aquí consideran el copy también.
    c. Líneas de conexión: cómo affordance lleva al usuario de un lugar a otro
6. Usan palabras para todo, no imágenes.

7. No se pierde tiempo en una innecesaria fidelidad o wireframes. Usan *fat marker sketches: grandes trazos. Nos permite segmentar nuestro propio proceso creativo sin entrar en mucho detalle.*
8. Los elementos son la salida (output)
9. Espacio para los diseñadores: cualquier mockup influenciará a la persona que venga detrás de ti. Moverse con el correcto nivel de abstracción asegura que nos movemos al ritmo correcto, y deja espacio para la creatividad en pasos posteriores. Si dejamos los detalles a un lado y usamos estas técnicas de breadboard y fat marker los diseñadores podrán tener más espacio para la creatividad en las siguientes fases del proyecto. Vamos haciendo el proyecto más específico y concreto pero dejando espacio.
10. La fase de shaping está más en una esfera privada. Hay todavía muchos puntos desconocidos y cosas que necesitamos ponerle dirección antes de que se pueda construir. El siguiente paso es testing para minimizar riesgo. Después de este paso ya se podrá pasar de shaped a preparado para el pitch.
11. En esta fase se puede dar por finalizado el proyecto. No se hacen promesas ni se compromete nadie. Lo que se hace es añadir valor a una idea en bruto haciéndola más accionable. Estamos llegando al punto de que sea una buena opción para asignar recursos.

Riesgos y “agujeros de conejo”

1. Tenemos una ventaja fija de tiempo. Tenemos que estar seguros que los elementos definidos sean ejecutables en 6 semanas.
2. Siempre habrá puntos desconocidos. Un proyecto shaped siempre debería estar libre de agujeros antes de considerarlo seguro para trabajar en él.
3. Los agujeros de conejo tienen que ver con puntos técnicos desconocidos, problemas de diseño no resueltos, o interdependencias que no se entienden. El proyecto tiene que tener partes bien entendidas, con independencia, partes que se pueden unir.
4. Una manera de descubrir estos agujeros es mediante un caso de uso en slow motion. Moviéndonos a cámara lenta podemos caminar del punto inicial al final -> revelando agujeros o piezas faltantes que necesitamos diseñar.
5. Cuestionar la viabilidad de cada parte.

    a. ¿Requiere esto trabajo técnico al que no nos hemos enfrentado antes?
    b. ¿Todo encaja como debería?
    c. ¿Estamos asumiendo que existe una solución de diseño a la que nosotros no podemos llegar?
    d. Decisiones difíciles: trade-offs que son difíciles dentro de un ciclo bajo presión. Hay muchas razones por las que un diseño diferente o una reconsideración de to-dos completados podría ser mejor.
6. Como shapers, se piensa más en la calidad básica y los riesgos que en el diseño definitivo. Con el concepto comprometido podemos mantener los elementos que hacen que merezca la pena.
7. Se muestran todos los casos que no soportas especialmente como parte del apetito (6 semanas).
8. Corta lo innecesario
9. Sal de la cueva del shaping para presentarle a los expertos técnicos las partes del concepto más técnicas si no estás completamente seguro. También puedes validar si los datos de uso están alineados con el supuesto del comportamiento del usuario que tú esperabas. Preséntalo como una idea. Está en fase shaping, recuerda.
10. En software todo es posible, pero nada es gratis. Queremos saber si es posible con el apetito que estamos shapeando. Pregunta: es posible sacar esto en 6 semanas?
11. Habla de las limitaciones con el equipo técnico. Enfatiza que estás buscando riesgos que podrían tumbar el proyecto. No es una conversación para ver qué piensa el otro. Invítalos a una habitación y dibuja los elementos en una pizarra. Construye el concepto desde el principio. ¿Tienen ellos sugerencias de cómo simplificarlo o un enfoque diferente al tuyo? Dependiendo de cómo vaya la conversación habrás validado tu enfoque o descubierto más problemas que te encaminarán hacia otra ronda de shaping.
12. Estamos listos para salir del shaping privado y recoger feedback. En la siguiente fase lo pondremos por escrito de una forma que comunique los límites y describa la solución para que las personas con menos contexto podrán entenderlo y evaluarlo. Este pitch será el documento que se usará para reclamar recursos, recolectar feedback en lo necesario o simplemente capturar la idea para cuando el momento sea más maduro en el futuro.

Escribir el pitch

1. Tenemos los elementos de la solución. Hemos minimizado el riesgo hasta el punto que confiamos que es una buena opción para presentar al equipo.
2. Lo ponemos por escrito para que los demás puedan entenderla, digerirla y responder a ella.
3. El propósito de un pitch es presentar una apuesta potencialmente buena. Es básicamente una presentación. Necesitamos capturar el trabajo hecho y presentarlo de una manera que el equipo que agende proyectos haga una apuesta informada.
4. Hay varios ingredientes que queremos incluir en un pitch:
    a. Problema: la idea en bruto, caso de uso, o algo que nos motive a trabajar en esto. Presentar la solución y el problema unidos. No des por hecho que es una buena idea crear algo porque sí o es atractivo para alguien. Establecer el problema ayuda a tener una conversación más clara cuando sea el momento de hacer el pitch o apostar por el concepto. Puede ser una solución perfecta sin un problema definido o que solo afecta a un porcentaje muy pequeño conocido que además tiene una retención baja. La mejor definición de un problema consiste en una historia específica que muestra por qué el status quo no funciona.
    b. Apetito: cuánto tiempo queremos invertir y cuáles son las limitaciones de la solución. 6 semanas. En caso de proyectos pequeños, 2 semanas. Queremos prevenir conversaciones que no sean productivas. Abrazar el apetito te convierte en un compañero de viaje. Lleva tiempo llegar a una idea simple que encaja en un espacio de tiempo tan reducido.
    c. Solución: los elementos núcleo o core presentados de una forma sencilla de entender
    d. Agujeros de conejo: detalles sobre la solución que valen la pena destacar para evitar problemas
    e. No-gos: cualquier cosa excluida del concepto: funcionalidades o casos de uso que nosotros intencionalmente no estamos cubriendo para encajar en el apetito o hacer el problema manejable.
5. No queremos problemas sin solución. Eso no está preparado ni para hacer pitch ni para apostarle. Darle esto a un equipo significa poner

investigación al nivel incorrecto donde el conjunto de sskills, limitaciones de tiempo y el riesgo están desalineados.

6. Solo se puede apostar por un proyecto que una problema, apetito y solución.
7. Cuando se escribe el pitch se hace con el nivel correcto de detalle. Hay que ser más concretos que en anteriores fases para que la gente que lo lea mire los dibujos y sin contexto pille la idea.
8. Se puede usar sketch para que quede más claro pero sin entrar demasiado en detalle. Lo que se quiere evaluar qué tan clara se presenta la funcionalidad por sí sola. Solo se usa sketch o algo para hacerlo visual en el caso de que se necesite vender bien el concepto de algo que no queda claro con otras técnicas (fat marker, breadboard, etc).
9. Algunas ideas son inherentemente visuales y otras muy difíciles de ver. Fat marker sketches pueden ser muy efectivos en un pitch. Hazlo limpio.
10. Puedes usar colores diferentes para diferenciar los labels de otras partes o puedes poner texto para que quede un elemento más claro.
11. WYSIWYG (What You See Is What You Get, "lo que ves es lo que obtienes"). Importante decir que lo que no está en el concepto no está considerado.
12. 2 capturas de pantalla demuestran el problema. Fat marker sketches describen la solución. Los agujeros de conejo motivan algunos sketches. Tamaño completo aquí: https://public.3.basecamp.com/p/5PbZ9Srki4NW6MUnFcbKPCh7
13. Este pitch para cambiar cómo las notificaciones funcionan: 2 vídeos para mostrar el problema. Un fat marker sketch y un breadboard describen la solución. Las cajas negras contienen las visualizaciones de datos que soportan los trade-offs de la solución. Tamaño completo: https://public.3.basecamp.com/p/NnuPFtsAEr7ZwixxmCjs7MwU
14. Listo para presentar: Comunicación asíncrona siempre, a ser posible. Tiempo real solo si es necesario. Comparten el pitch con todos los ingredientes donde todos puedan verlo. En condiciones ideales todos tienen tiempo para leer el pitch con antelación. Si no es posible en algunos casos, el pitch está preparado para presentar sin contexto de los demás.
15. En Basecamp postean los pitches como *Messages* en Basecamp. Crearon una categoría llamada Pitch. Los pitches son publicados en un equipo llamado Product Strategy que es accesible por la gente en la “betting table”.

16. Cuando necesitan incluir un fat marker sketch en un pitch, lo dibujan con iPad con Notability y toman una captura. Lo adjuntan en Basecamp directamente. La gente comenta el pitch de manera asíncrona. No dicen sí o no, eso es parte de la betting table. Los comentarios contribuyen a tapar agujeros.

Parte 2 - Betting

1. En este capítulo veremos el proceso en detalle para ver dónde van los pitches y cómo convertirlos a proyectos agendados.

Di no a los backlogs

2. Fuera backlogs. Cientos de tareas para las que nunca tendremos tiempo. No queremos tener un sentimiento de que cargamos esta pila. Son una pérdida de tiempo porque tienes que estar constantemente revisando, grooming, etc.
3. En lugar de backlogs, antes del ciclo de 6 semanas, se reúnen con la mesa de betting para decidir qué foco tomar. Se miran los pitches de las últimas 6 semanas. Y nada más. Están los pitches bien shapeados, reducidos al mínimo riesgo, los que son apuestas potenciales.
4. Reunión productiva, infrecuente y corta. Pocas opciones y un ciclo de seis semanas. Si no lo ven. Se deja ir. Si lo ven pero no es el momento, alguien lo puede proponer para el siguiente ciclo.
5. Listas descentralizadas por equipos con prioridades. No hay un backlog centralizado. Soporte puede hablar con producto de top issues y producto puede plantearlo como proyecto para shaping. Este enfoque permite que sea manejable y que exista responsabilidad y priorización. La conversación se mantiene fresca porque si hay que volver en un asunto se tiene contexto, y propósito. Cada cosa es relevante y timely.
6. Las ideas no valen nada. Es fácil sobrevalorarlas. Se acumulan en grandes colas. Las más importantes volverán. Cuando se repita el problema estarás motivado por shapear una solución y darle forma al pitch para apostarle en el siguiente ciclo.

La mesa del betting o de la apuesta

1. Ahora que tenemos potenciales apuestas en la mesa en forma de pitches, tiempo de tomar decisiones de qué proyectos agendar.

2. Ciclos de seis semanas: sprints de dos semanas es muy poco para sacar algo significativamente bueno. Ciclos de dos semanas son costosos por el planning. La cantidad de trabajo que es posible sacar en dos semanas no vale la pena para el tiempo que se pasa con el sprint plan o el coste de oportunidad de romper el momentum. Ciclos de seis semanas son suficientemente largos como para terminar un proyecto de inicio a fin. Se puede ver el final desde el principio. Si el deadline es demasiado abstracto el equipo usaré el tiempo de manera ineficiente.
3. El final de un ciclo es el peor momento para reunirse y planificar porque todo el equipo está estresado por sacar el proyecto a tiempo. Después de las seis semanas, reservamos dos para relajarnos sin trabajo agendado, tiempo para respirar, reunirse lo que sea necesario y considerar lo que viene.
4. Durante estas dos semanas, diseñadores y programadores están llamados a trabajar en lo que consideren. Ellos disfrutan este tiempo después de 6 semanas trabajando duro en el proyecto. Suelen resolver bugs, explorar nuevas ideas o intentar nuevas posibilidades técnicas.
5. Equipos: 1 diseñador / 2 programadores o 1 programador / 1 diseñador. Se une una persona de QA que actúa más adelante en la etapa.
6. 1 proyecto para el ciclo o múltiples para el ciclo. Big batch vs. small batch. Small: 1 o 2 semanas.
7. The betting table: apuestas potenciales shapeadas (pitches) durante las últimas 6 semanas. No hay backlog que organice. Quién es parte? CEO (última palabra en producto), CTO, programador senior y Ryan (estrategia de producto). Que te lo compren desde arriba es fundamental para que los ciclos funcionen. El output de la llamada con el C-level es el plan del ciclo. No se pierde el tiempo. Los que “mandan” forman parte de la betting table. Pocas opciones shapeadas.
8. El pitch define un payout específico que hace que la apuesta valga la pena. Las apuestas son compromisos. No puede haber interrupciones por 6 semanas en el equipo. Cuando se interrumpe por un bug se rompe el *momentum* y perder esa hora te puede arruinar el día completo. Si hay crisis claro pero no suele pasar
9. Solo 1 apuesta por ciclo. Si no se acaba no hay extensión. Elimina el riesgo de dejar los proyectos pasar. Si no se termina en 6 semanas es que se ha hecho mal el shaping. Más *ownership* de los proyectos. El equipo tiene que decidir qué decisiones afectan el alcance.

10. Los bugs no interrumpen al proyecto a no ser que sean críticos. Pero esto es raro. La gran mayoría pueden esperar. 3 estrategias para lidiar con ellos:
    a. Periodo concreto para enfocarse en bugs
    b. Llévalo a la betting table. Hay una gran diferencia entre retrasar un ciclo para resolver un bug vs. decidirlo antes y dedicar tiempo para resolverlo.
    c. Dedica un ciclo completo a resolver bugs (tirando a vacaciones). Es el momento perfecto porque hay muchos que toman vacaciones.
11. Cada 6 semanas aprendemos qué funciona y qué no; qué importa y qué no.
12. Qué pasa con los proyectos que no se pueden sacar en 1 ciclo? Da igual. Solo reservamos 6 semanas. Si no se acaba, se define otro proyecto. Solo se shapea lo que parece que tiene un principio y final en 1 ciclo y con eso mantenemos opciones de cambiar de rumbo.

Place your bets

1. Mejoramos productos existentes o creamos nuevos productos? Si mejoramos productos existentes seguimos el proceso de Shape Up estándar: shapear el trabajo, apostarle, dárselo al equipo para que lo haga realidad. Se espera que se entregue una versión terminada del trabajo shapeado al final del ciclo.
2. Crear nuevos productos desde 0 es otra cosa. 3 fases:
    a. modo R&D: en shaping aprendemos qué queremos construyéndolo. En R&D no se apuesta por un pitch shapeado, principalmente se tiene en cuenta los avances en UI o piezas de código que se pueden usar a futuro. Los senior hacen el equipo. No puedes delegar algo que no sabes ni tu mismo. Se necesita juzgar el impacto de las decisiones en el largo plazo. No se espera lanzar nada al final de un ciclo de R&D. El objetivo es ver qué funciona: el código principal y la UI definirán la forma del producto. No más de un ciclo de R&D.
    b. modo producción: cuando las decisiones importantes de arquitectura están hechas, el producto hace las cosas esenciales que lo definen, el equipo senior puede traer a otras personas para contribuir. El trabajo precedente de R&D sirve para que los nuevos contribuidores identifiquen donde la nueva funcionalidad va y cómo encaja con todo lo demás. En este punto trabajamos en ciclos

formales con shaping, betting y building. Shaping es deliberado en este punto. Establece qué se quiere ver al final del ciclo. El equipo que construye no está limitado por un grupo senior. Es posible trabajar con varios equipos a la vez. Shipping (lanzar) es el objetivo ahora. Shipping no es lanzar al público sino mergear a una base de código y esperar que no se vuelva a tocar. Se mantiene la opción de eliminar funcionalidad antes de lanzar. Se puede experimentar.

   c. modo limpieza: Siempre hay cosas que se olvidan, o detalles que se pierden o bugs que aparecen después de las fases de R&D y producción. Se reserva un tiempo al final para lo inesperado. En el modo limpieza: no hay shaping, no hay limitaciones de equipo claras, el trabajo está lanzándose continuamente. La disciplina es importante aquí. La limpieza no puede ser más larga que dos ciclos. Este modo es el momento donde el liderazgo toma decisiones finales de cortar algo. A veces hay que ver las cosas como un todo para saber con lo que podemos vivir sin y lo que requiere más consideración antes de lanzar la v1: responder a las preguntas, soporte al cliente, mantenimiento…

3. Ejemplos: The Dot Grid Calendar (en Basecamp, producto existente). HEY (nuevo producto) -> 2 AÑOS de desarrollo, modo R&D por el primer año de su desarrollo, 1 año de modo producción antes de configurarlo junto al núcleo. 2 ciclos de limpieza antes de lanzar en julio 2020. Ponen otro ejemplo que no esperaban lanzar al público, shapearon una versión suficientemente funcional. Se sintieron seguros después del 1er ciclo. Decidieron finalmente después de otro ciclo lanzarlo.
4. Preguntas para hacerse en la betting table:
   a. ¿Importa el problema? La solución no importa si el problema no vale la pena. Claro, cualquier problema que afecta a los clientes importa, pero necesitamos tomar decisiones y seleccionar porque siempre habrá más problemas que tiempo para resolverlos. Juzgan el problema desde su perspectiva, puesto y conocimiento.
   b. ¿El apetito es correcto? Está correcto cuando tenemos una solución shapeada en un tiempo razonable, como dos o seis semanas.
   c. ¿Es la solución atractiva? El problema vale la pena, el apetito es justo, pero ¿la solución es atractiva? Puede resolver el problema pero no ser adecuada la solución y puede causar problemas en el

futuro. No se discuten soluciones técnicas o trabajo de diseño en la betting table, es más alto nivel.

5. ¿Es el momento oportuno? El proyecto puede estar bien, pero puede que no sea el momento adecuado para afrontarlo.
6. ¿Están las personas correctas disponibles? Como parte del proceso de betting, se eligen los roles para cada equipo. Se elige de un pool de diseñadores y programadores (core product) cuando se planea para cada ciclo. Diferentes proyectos requieren diferente experiencia. Puede ser que alguien que haya hecho small batch quiera ahora un big batch y viceversa. Hay que tener en cuenta la disponibilidad: vacaciones, días sabáticos…
7. Después de decidir las apuestas o bets, alguien de la betting table (Jason, DHH, alguien senior) escribe un mensaje anunciando a todo el mundo a qué proyectos vamos a apostarle para el siguiente ciclo y quiénes van a trabajar en cada uno de ellos.

Parte 3 - Building

Responsabilidad

1. Tiempo de empezar un nuevo ciclo después de las apuestas: ¿por dónde empezamos?
2. Asignamos proyectos, no tareas. Nunca perdemos de vista la foto general. Confiamos en el equipo de controlar el proyecto completo con las limitaciones descritas en el pitch. El equipo define las tareas y el enfoque de cara al trabajo. Tienen autonomía total para ejecutar el pitch como mejor les parezca. La gente talentosa no soporta que los traten como monos o ejecutores de tickets.
3. Lo que funciona en papel rara vez funciona en digital. Los diseñadores y programadores que hacen el trabajo están en la mejor posición para juzgar cómo todas las piezas se conectan.
4. No es darle al equipo completa libertad para trabajar en una solución desde 0. Hemos hecho shaping. Hemos establecido las limitaciones. Ahora el equipo implementa y determina cómo ejecutarlo desde la perspectiva de diseño y desarrollo. Esto es lo que se llama el correcto nivel de abstracción.
5. Done means deployed: hecho significa desplegado en prod. Al final del ciclo. El proyecto necesita ser completado en el tiempo establecido. De

otra manera, el apetito no significaría nada. Cualquier tarea de testing o QA necesita que pase durante el ciclo. No somos estrictos en lo que se refiere al timing de la documentación, marketing updates, anuncios, etc. No se esperan dentro del ciclo.

6. Kick-off: empezamos el proyecto creando un nuevo proyecto en Basecamp y añadiendo al equipo completo. Lo primero que se hace es publicar el concepto shapeado en el muro de Mensajes. Se puede pasar el pitch original o una versión.
7. Todo es remoto. Importante usar bien el chat. Se organiza una llamada donde salen preguntas que surgen a partir del doc. Con el entendimiento completo del pitch empiezan a trabajar.
8. En los primeros días del proyecto trabajar no es trabajar es encontrar orientación. Se comprueba cómo funciona el sistema y qué punto de partida es el mejor. Todos están ocupados aprendiendo, nada se despliega, o no se cierran tareas. Es importante que los managers respeten esta fase. Las interferencias o preguntar temprano por los entregables chafa todo. Toma tiempo encontrar el enfoque correcto.
9. Tasks que realmente necesitan hacerse vs. tasks que pensamos que deberían hacerse. Los detalles inesperados presentan muchas veces los retos más complejos. Los equipos descubren las tareas haciendo trabajo real. No se trata de ponerse a escribir cualquier cosa sino algo significativo de principio a fin en unos días, sea cosa de UI o de código.

Get one piece done

1. Tan pronto como el equipo se oriente, toca ponerse con las tareas pero no sueltas, tienen que terminar en una cosa concreta. Es difícil seguir el proceso si no. Puedes sentir que has hecho muchas tareas pero no puedes mostrar nada significativo y eso no es sentir que estamos progresando. Todo está hecho pero nada está hecho. Necesitamos algo tangible en la primera semana.
2. Dos capas: frontend y backend. Diseño y código. Técnicamente hablando hay más capas que estas dos pero estas son las primarias en los retos de los proyectos, sobre todo, porque el diseño afecta al backend y el backend afecta al frontend y viceversa.

3. Los programadores no necesitan esperar por los diseños cuando empiezan: otra ventaja del proceso shapping. Hay suficiente dirección en el pitch para empezar a trabajar en los problemas del backend desde el principio.
4. Los programadores no necesitan diseños perfectos para empezar a implementar. Lo que necesitan son endpoints: input elements, botones, lugares en los que la data almacenada debería estar. Estos affordances son el núcleo del diseño de UI.
5. Cuestiones fundamentales: tiene sentido? se entiende? hace lo que queremos?
6. La primera interfaz que un diseñador le da a un programador es básica: diseño visual o muck-up. Esa parte visual es importante al final, no en etapas iniciales. Las grandes preocupaciones están en si funcionará, en si hará sentido y en lo duro que será implementarlo. Primero hazlo funcional, luego hazlo bello. Lo mismo pasa con el backend. No tiene que ser todo o nada.
7. Los criterios para saber qué construir primero:
    a. Debería ser core. Lo central e importante que haga que el proyecto se pruebe temprano en el ciclo.
    b. Debería ser pequeño. Terminar algo significativo en los primeros días y construir *momentum* para que el equipo sienta que estamos en el camino correcto.
    c. Debería ser original para atraer la confianza de los demás.

Map the scopes (alcance)

1. Organiza por estructura, no por persona o rol. No queremos que el equipo complete tareas, queremos terminar el proyecto - cosas que pueden hacerse o acabarse. Cuando te adentras en el trabajo de desarrollo de producto, descubres interdependencias, cómo se conecta todo, y lo que puedes dejar de lado. Integrar frontend y backend es clave.
2. Las tareas son el punto de partida natural porque son concretas y granulares. Sería artificial categorizarlas arbitrariamente. Es suficiente con capturar al principio un conjunto de cosas que deberían pasar. Cuando el equipo empieza a meterse en trabajo “real” es capaz de establecer alcances. Es como dividir el mapa del proyecto en territorios independientes. Son partes importantes del problema que pueden ser completadas independientemente y en un corto periodo de tiempo - pocos días. El mapa sirve como imagen mental. En la práctica, definimos y

medimos los alcances como To-do lists. Cada scope, la lista de un nombre. Cualquier tarea para ese scope va en esa lista.

3. Los alcances son como el idioma del proyecto en un nivel macro. Nos interesa poder ver las partes más importantes como alcances.
4. Para hacer el mapa de los alcances tienes que caminar el territorio. Los alcances bien hechos reflejan la verdad de lo que se puede crear independientemente. Necesitamos descubrirlos a medida que vamos avanzando.
5. Sabemos que el alcance es correcto cuando:
    a. Podemos sentir que vemos el proyecto completo y no nos dejamos nada importante en los detalles
    b. Las conversaciones empiezan a fluir porque los alcances te dan el idioma o lenguaje correcto
    c. Cuando salen nuevas tareas, sabes dónde ponerlas.
6. 3 señales indican que los alcances deberían rehacerse:
    a. Es difícil decir lo “hecho” que está un scope
    b. El nombre no es único para el proyecto.
    c. Es demasiado grande para acabarlo pronto.
7. Layer cakes: la mayoría de los proyectos de software requieren diseño y código debajo. A veces la balanza no está equilibrada y hay mucho más trabajo de un lado que de otro. Es la complejidad necesaria realmente o se puede reducir? necesitamos una UI tan fancy? Hay una manera diferente de construir ese backend y tener menos interdependencias con el resto de los sistemas?
8. Hay cosas que no entran en los scopes. Se crea una lista excepcional (miramos escépticos) para tareas que no encajan en ningún scope.
9. Marcar los nice-to-have con un símbolo. Esto ayudará al equipo a ordenar los must-to-haves de los nice-to-haves. No hay que olvidar que tenemos un periodo de tiempo fijo.

Mostrar progreso

1. Los managers tendrían que poder ver directamente cómo va el proyecto sin preguntar status. Necesita contexto desde dentro del scope para entender qué significa que esas tareas están hechas y otras podrían llegar. Las estimaciones tienen un problema: diferentes naturalezas en función del trabajo que estimes. Sin experiencia, difícil o con interdependencias también complicado. Los “unknowns” son determinantes. La manera que

encontramos en Basecamp de ver el status sin contar las tareas ni hacer estimaciones: metáfora del hill chart. Básicamente: foco en qué está hecho, qué no, qué es desconocido, qué está resuelto.

2. 2 fases de cada trozo del trabajo: conocido y desconocido. Uphill: qué vamos a hacer y el enfoque: fase downhill de ejecución. El hill chart muestra cómo se ve el trabajo en diferentes etapas. La fase de uphill está llena de incertidumbre, desconocidos, y resolución de problemas. La fase de downhill está marcada por la certidumbre, confianza, verlo todo y saber qué hacer.
3. Scopes en el hill chart: Los alcances nos dan el lenguaje del proyecto (locate, reply…) y el hill describe el status de cada alcance o scope (uphill o downhill). Podemos poner diferentes colores en el hill para ver el status. Para los managers la habilidad de comparar estados pasados es la funcionalidad clave. No solo puede observar dónde estamos sino cómo nos estamos moviendo. De esta manera, pueden juzgar qué problemas elige el equipo para resolver y cuánto tiempo ellos invierten en cada etapa desde los desconocido pasando por lo conocido hasta lo hecho.
4. Nadie dice “no sé cómo solucionarlo”. Con este sistema no acumulamos riesgos y eliminamos la incertidumbre. Un punto que no se mueve, mala señal. Es menos alrededor de la persona, y más alrededor del trabajo en sí. Cómo podemos resolverlo para superar el hill?
5. No hagas el trabajo con la cabeza, hazlo con las manos. No te emociones moviendo el punto, básate en la realidad. Pasa de idea, a validación, a creación o ejecución -> seguro que no hay más desconocidos.
6. Resuelve el trabajo en la secuencia correcta. Se puede usar el hill chart para saber qué problemas resolver en qué orden.
7. Algunos scopes son más arriesgados que otros. Con más desconocidos. Es crítico superar el hill de los más retadores y dejar lo demás para el final. Los problemas más importantes primero con lo más desconocido y deja las cosas que sean más rutinarias para el final. Al final del ciclo, el equipo habrá terminado lo importante y dejado una variedad de nice-to-haves y de maybes por fuera.

Cuándo parar

1. Siempre hay más trabajo que tiempo disponible. Lanzar en el tiempo programado quiere decir lanzar algo imperfecto. Es suficientemente bueno? Está preparado para lanzarse?
2. Los diseñadores y programadores siempre quieren hacer su mejor trabajo. Orgullo en el trabajo es importante para la calidad y la moral, pero necesitamos ir al objetivo correcto. No será perfecto. Tampoco podemos bajar los standards. Cómo sabemos que lo que tenemos es ya bueno para movernos a otra cosa? Comparando. No con lo ideal, sino con la realidad de los clientes. Cómo resuelven el problema los clientes hoy sin esta funcionalidad? Qué frustración estás eliminando? Cuánto más tienen que esperar los clientes para recibir la solución mientras nosotros decidimos si diseño a o b?
3. Viendo cómo nuestro trabajo es mejor que las alternativas actuales nos hace sentir mejor sobre el progreso hecho. No es tanto por nosotros, sino por el valor entregado a los clientes. Es la diferencia entre “no es suficientemente bueno” y “mejor de lo que ya teníamos”. Podemos decir que no es perfecto pero el cliente definitivamente piensa que es una gran mejora respecto a lo que tenía.
4. Los límites motivan los trade-offs. Hay tiempo para eso? Sin un deadline, podríamos retrasar el proyecto por cambios que no merecen tiempo extra. En basecamp esperan que sus equipos hagan trade-offs constantemente y se pregunten si el scope les empuja a terminar sus tareas.
5. No puedes ver los detalles hasta que te metes hasta dentro en el trabajo. El scope no para de crecer. Es natural. No solo descubrirás complejidades no anticipadas sino cosas que podrías arreglar o hacer mejor. No intentemos parar el scope sino darle al equipo más herramientas, autoridad y responsabilidad para rebajarlo. Cada proyecto está lleno de scope que no es necesario. Cada parte de un producto no necesita ser igual de prominente, rápida, pulida. Cada caso de uso no es igual, igual de crítico o alineado de la misma manera con el mercado que estamos atacando.
6. Rebajar o cortar el scope no es bajar el estándar de calidad. Tomar decisiones hace el producto mejor. Diferenciar lo core de lo periférico nos mueve a ser competitivos en nuestra industria o espacio, haciéndonos más parecidos o diferentes que otros productos que tomaron otras decisiones. Por lo tanto, tener un scope variable no tiene que ver con

sacrificar calidad. La cosa es preguntarse qué importa realmente, qué cosas nos empujan hacia adelante y qué cosas nos diferencias para los casos de uso que intentamos resolver.

7. Preguntarse:
    a. Es una funcionalidad must-have?
    b. Podemos lanzar sin esto?
    c. Es un nuevo problema o ya existía?
    d. Cómo de frecuente es este caso?
    e. Cuando ocurre, quién lo ve? Es core?
    f. Cuál es el impacto de este caso?
    g. Qué tan alineado está este caso de uso con mi audiencia?
8. El deadline fijo nos motiva a hacernos preguntas. El scope variable nos permite actuar. Enfoque en algo que nos haga estar orgullosos al final de las 6 semanas o del ciclo que sea.
9. En Basecamp tienen 1 persona de QA. Viene al final del ciclo y “caza” los casos fuera de la funcionalidad core. Puede limitarse a estos casos puntuales porque los diseñadores y programadores son responsables en su lado asegurando la calidad básica de sus tareas. El equipo tiene que asegurarse que el proyecto se lanza alineado con lo que se había planteado en el proceso shaping. Esto nos devuelve a la idea de asignarle el equipo o empoderarlo con el proyecto completo y no solo con tareas individuales.
10. No dependen de un QA para lanzar funcionalidades de calidad pero son mejores con QA.
11. Los issues que vienen de QA se meten en una lista separada de to-dos. Si el equipo decide que ese issue es must-have, entonces lo llevan a la lista del scope que sea relevante. Esto ayuda al equipo a ver que el scope no está terminado hasta que el issue haya sido atajado. Tratan el código de la misma manera. El equipo puede lanzar el código sin code reviews. No es formal. Pero la code review hace el código mejor. Hay tiempo y hace sentido que un senior lo vea y pase feedback. Es más aprovechar la lección que puede ser y la oportunidad de aprender de otros más experimentados a crear un paso extra en el proceso.
12. Cuándo extender un proyecto? En situaciones contadas, se extienden dos semanas pasado el deadline. Las tareas que quedan tienen que ser verdaderos must-haves. Todo el trabajo restante tiene que estar en la fase downhill. No queremos problemas sin resolver o preguntas abiertas. Los desconocidos son demasiado arriesgados para apostar por ellos más

tiempo. Si hay mucho en el uphill ya se considerará más adelante. A veces, aunque las condiciones permiten extender, lo mejor es ser rigurosos con el ciclo y disciplinados para respetar el apetito. No puede convertirse en un hábito el extender.

Move on

1. Deja que pase la tormenta. Lanzar algo a producción puede generar más trabajo si te descuidas. Bugs salen, las peticiones de los usuarios por cosas nuevas crecen… Todo el mundo reacciona. El feedback puede ser especialmente intenso si la funcionalidad cambia flujos de trabajo existentes. Es importante no reaccionar como un cretino y mantenerse de buen rollo. Darle unos días, que se les pase. Sé firme y recuerda por qué has hecho el cambio y a quiénes les estás ayudando.
2. Permanece sin deuda. Decir que NO no quiere decir no continuar contemplándolo y hacer shaping en el futuro. Decir que sí te compromete para el futuro. Recuerda, lo que has lanzado lo has trabajado por seis semanas. Si necesita más tiempo, necesita una nueva apuesta. Deja que las requests o los bugs que lleguen hablen y en la siguiente betting table ya decidirán.
3. El feedback necesita ser shapeado: Las ideas que vienen de los usuarios pueden no ser accionables aún. Necesitan shaping. Son inputs en bruto. Si una solicitud es importante de verdad, puedes hacer que sea prioridad para el siguiente ciclo. Tiene que estar bien shapeada para llevarla a al betting table.

Conclusión

1. Podría llevar tiempo y experimentación implantar la metodología Shape Up en tu equipo. Adáptala y adóptala a tu manera. Conceptos importantes:
    a. Shaped vs. Unshaped
    b. Establecer apetitos en lugar de estimaciones
    c. Diseñar con el nivel correcto de abstracción
    d. Los conceptos con breadboards y sketches
    e. Apostarle sin interrupciones
    f. Seleccionar la duración correcta del ciclo
    g. Periodo de recapacitación después del ciclo
    h. Partir los proyectos en scopes
    i. Downhill vs. uphill

j. Comunicar lo desconocido
k. Separar must-haves de nice-to-haves

Parte 4 - Appendices (How to Implement Shape Up in Basecamp | Shape Up)

1. Cómo implementar Shape Up en Basecamp
2. Ajustar Shape Up al tamaño de tu compañía
3. Cómo empezar a trabajar con Shape Up
4. Glosario
5. Sobre el autor

FIN.