Green Coding

El cambio climático es uno de los mayores retos a los que se enfrentará la humanidad durante las próximas décadas. Las empresas de tecnologías de la información y la comunicación (TIC) pueden marcar la diferencia con GreenCoding.

Motivación

El 20 de enero de 2021, el nuevo presidente de Estados Unidos, Joseph R. Biden Jr., acababa de ser investido y quiso tomar medidas contra el cambio climático, prometiendo 1,7 trillones de dólares de inversiones en energía limpia y emisiones netas cero en Estados Unidos para 2050.

Mientras tanto, al otro lado del charco, la Comisión Europea se ha comprometido a destinar 100.000 millones de euros en inversiones para la transición a la neutralidad climática en el mismo periodo, con el fin de alcanzar objetivos políticos y compromisos como el Acuerdo Climático de París en la COP 21.

Por lo tanto, es evidente que la reducción de las propias emisiones de CO2e se convertirá en una prioridad aún mayor para las empresas y asociaciones a nivel mundial.

Índice

Además, el crecimiento de la inversión en fuentes de energía renovable y en tecnología verde, como los coches eléctricos, ilustra que, si bien estas áreas están ganando significativamente impulso, siguen siendo incipientes durante estos años; especialmente porque en 2019, sólo el 11% de la energía primaria del mundo provenía de fuentes renovables. Por lo tanto, la prioridad debe ser demostrar la reducción de las emisiones a través de la innovación de los procesos de negocio y la cadena de valor, adoptando un enfoque de evaluación del ciclo de vida que garantice la reducción del consumo de energía y recursos en general, y no sólo la disminución de las emisiones de CO2. Analizar los procesos principales en toda la cadena de valor significa que, con el tiempo, al aplicar cambios graduales, se podrá lograr una reducción sustancial no sólo de las emisiones y el uso de recursos, sino que además se obtendrán ganancias sustanciales en la mejora de la eficiencia general.

En la actualidad, la mayoría de las empresas sólo se centran en sus emisiones directas, conocidas como emisiones de alcance 1 y 2 según el Protocolo de Gases de Efecto Invernadero (GHG Protocol, por sus siglas en inglés). Éstas se deben principalmente a determinados procesos de producción de bienes, como la generación de frio y calor. Pero para muchas organizaciones el mayor impacto se deriva de las emisiones indirectas (alcance 3), también denominadas emisiones de la cadena de valor, como las relacionadas con el uso real de los productos[1].

Las emisiones de alcance 3 son especialmente relevantes en el sector de las tecnologías de la información, ya que las emisiones derivadas del desarrollo sólo representan una pequeña parte. Utilizando a Microsoft como ejemplo, de un total de 16 millones de toneladas métricas de emisiones de carbono en 2020, alrededor del 75% corresponden al alcance 3[2]. En general, la programación siempre tiene que ver con la eficiencia del esfuerzo; pero ¿tienen los desarrolladores realmente en cuenta la eficiencia energética cuando codifican?

Estudios recientes demuestran que la demanda de electricidad en los ámbitos de las tecnologías de información y la comunicación representa actualmente entre el 5% y el 9% de la demanda mundial de electricidad, y las previsiones indican que esta cifra podría aumentar hasta el 21% en 2030 [3,4].

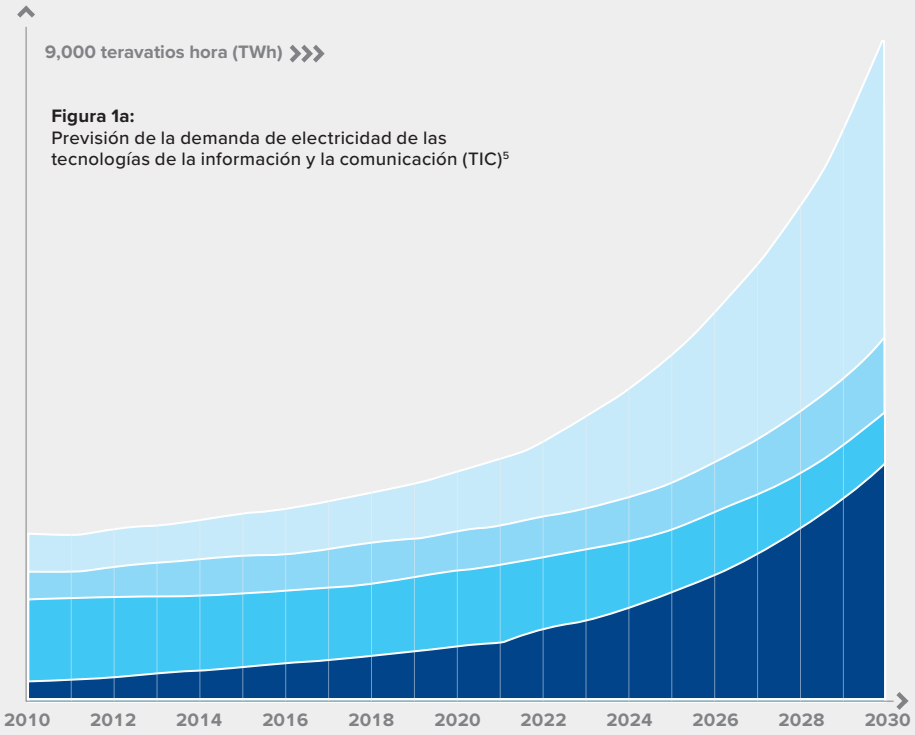


Figura 1a:
Previsión de la demanda de electricidad de las tecnologías de la información y la comunicación (TIC)[5]

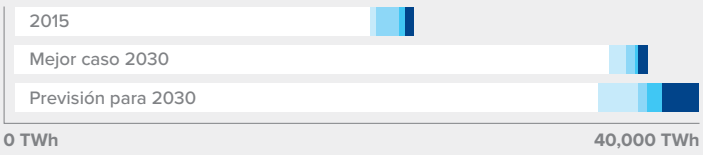


Figura 1b:
Demanda mundial de electricidad

[1] WWF Germany, "Overcoming barriers for corporate scope 3 action in the supply chain", https://www.wwf.de/fileadmin/fm-wwf/Publikationen-PDF/WWF-Overcoming-barriers-for-corporate-scope-3.pdf, December 2020

[2] Microsoft "Microsoft will be carbon negative by 2030", https://blogs.microsoft.com/blog/2020/01/16/microsoft-will-be-carbon-negative-by-2030/, September 2020

[3] Huawei Technologies Sweden AB, "On Global Electricity Usage of Communication Technology: Trends to 2030": https://www.mdpi.com/2078-1547/6/1/117, April 2015

[4] Enerdata, "Between 10 and 20% of electricity consumption from the ICT sector in 2030?": https://www.enerdata.net/publications/executive-briefing/between-10-and-20-electricity-consumption-ict-sector-2030.html, August 2018

[5] Springer Nature, "How to stop data centres from gobbling up the world's electricity": https://www.nature.com/articles/d41586-018-06610-y, September 2018

Por qué las empresas tienen que ser conscientes de este tema - y por qué los programadores deberían pensárselo dos veces antes de codificar

La concienciación pública sobre las posibilidades que ofrece el desarrollo de software sostenible es mínima, aunque pioneros como Alex Russell y Jeremy Wagner [6,7] llevan tiempo intentando cambiar esta situación. Para marcar la diferencia, será necesario sensibilizar a todos los ámbitos y a todas las partes interesadas: empresas, proveedores, consumidores y creadores.

Desde el punto de vista de negocio o de management, esta escasa concienciación sobre el consumo energético del software se debe principalmente a un factor: los presupuestos de desarrollo y operativos están en su mayoría “desacoplados”. Como resultado, existe un evidente conflicto de intereses. Por un lado, hay que invertir más tiempo de desarrollo para llevar a cabo pruebas de eficiencia, pero por otro lado, los costes operativos aumentan cuando se trabaja con menos eficiencia. Por tanto, los managers deben centrarse en la sostenibilidad como resultado ideal. Deben dejar de lado la prioridad tradicional, que se centraba exclusivamente en el rendimiento del desarrollo y la reducción de costes, y centrarse en cambio en una prioridad general: optimizar el software de principio a fin.

Obviamente, el equipo directivo es sólo la primera capa en este cambio de mentalidad sugerido. Los analistas, arquitectos e ingenieros de software también deben comprometerse si las empresas desean embarcarse en este viaje, partiendo de la premisa de que los ordenadores son simplemente máquinas, como cualquier otra. La eficiencia energética depende del software que ejecutan las máquinas. Todo código crea una huella de carbono, por lo que es responsabilidad de todos nosotros garantizar que esta huella sea lo más pequeña posible.

Si pensamos en esta cuestión en términos de rendimiento final a escala global -con los proveedores cloud operando continuamente los servidores de la infraestructura, trabajando junto a otros proveedores y empresas- hay un potencial considerable para ahorrar energía. Sin embargo, hay que tener en cuenta que esto no sólo se aplica a las aplicaciones convencionales, como los sistemas operativos, la tecnología de ofimática o las aplicaciones de servidor. Si se amplía a cientos, miles o incluso millones de dispositivos (ordenadores de sobremesa, teléfonos inteligentes, tabletas...), cada pieza de código puede contribuir de forma importante a reducir el consumo de energía, ayudando así a reducir las emisiones globales de CO2.

[6] “Alex Russell – The Mobile Web: MIA”: https://vimeo.com/364402896, October 2019
[7] “Responsible JavaScript” by Jeremy Wagner: https://speaking.jeremy.codes/Vci5ad/responsible-javascript, October 2019

El alcance y potencial de GreenCoding

Aquí es donde entra en juego el denominado GreenCoding. La idea es programar, desarrollar y ejecutar el software de una forma mucho más respetuosa con el medio ambiente.

Sphexishness [sfɛksɪʃnəs] es un término acuñado por Douglas Hofstadter para referirse a estar atrapado en la rutina del pensamiento automático

Un requisito previo clave para el GreenCoding es repensar, reconsiderar, las cosas. Esto implica adoptar un enfoque holístico para todas las cuestiones o problemas de negocio y, en la medida de lo posible, evitar el pensamiento automático arraigado, o "sphexishness".

GreenCoding comienza con la planificación de un proyecto cuando se analizan los requisitos iniciales. En esta fase, la prioridad fundamental es seleccionar una plataforma adecuada para el proceso de desarrollo. Algunos estudios han descubierto que algunos lenguajes de programación ya tienen un gran impacto en la eficiencia energética y la velocidad; permitir a los programadores elegir un lenguaje de programación adecuado puede suponer una enorme diferencia en lo que respecta al ahorro de energía y el rendimiento. Hay que tener en cuenta que esto es sólo un ejemplo, dependiendo del proyecto hay que entender qué decisiones tendrán un mayor impacto.

En última instancia, GreenCoding significa añadir una nueva pregunta al proceso de diseño. Los equipos deben preguntarse si hay una forma mejor de obtener el beneficio deseado con la menor cantidad de energía gastada posible. Responder a esta pregunta puede tener un gran impacto en el diseño. Por ejemplo, podría conducir a un enfoque "serverless" para optimizar la infraestructura, o podría decidir adaptar la experiencia del usuario para minimizar el tiempo invertido por los usuarios finales del software.

TOTAL

ENERGÍA		TIEMPO		MB	
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Figura 2:
Energía, tiempo y memoria requeridos por los distintos lenguajes para un producto de referencia [8].

[8] Energy efficiency by programming language: https://greenlab.di.uminho.pt/wp-content/uploads/2017/10/sleFinal.pdf, October 2017

Hay tres pilares importantes que hay que tener en cuenta a la hora de escribir software:

Estos tres pilares afectan al código de diferentes maneras, y cada uno de ellos debe tratarse por separado. Sin embargo, la primera tarea es sentar las bases de las que dependen estos pilares y establecer una visión general de alto nivel antes de implementar los mecanismos del proceso de desarrollo.

Lo "QUÉ"

se genera (el propio código). ¿Es eficiente en términos de beneficio entregado o de energía invertida?

UNA LÓGICA MÁS VERDE

"CÓMO"

se genera. ¿Es eficiente el ciclo de vida de desarrollo del software? ¿Podría generarse el mismo código con menos energía?

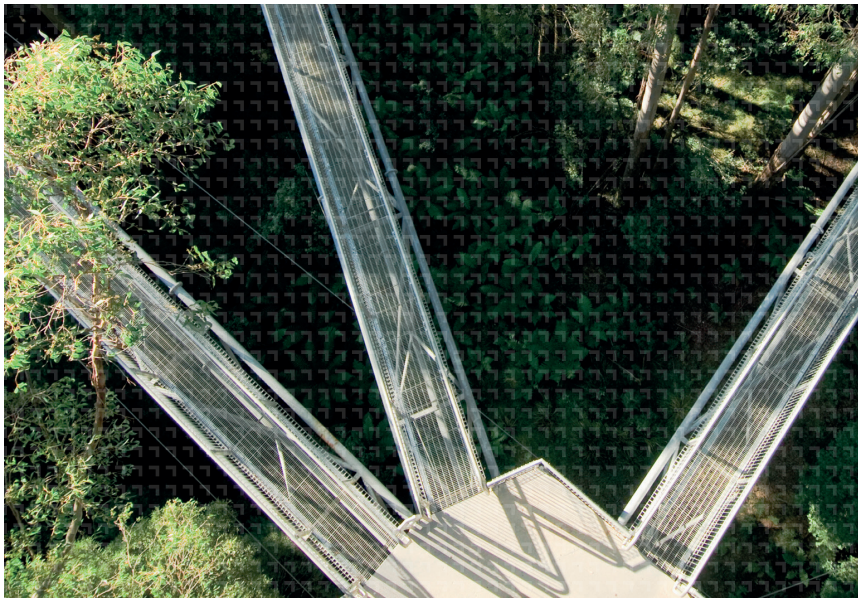
UNA METODOLOGÍA MÁS VERDE

"DÓNDE"

se ejecuta el software (la plataforma final que ejecuta el código). ¿Consume la mínima energía necesaria para ejecutar el código generado?

UNA PLATAFORMA MÁS VERDE

ARQUITECTURA

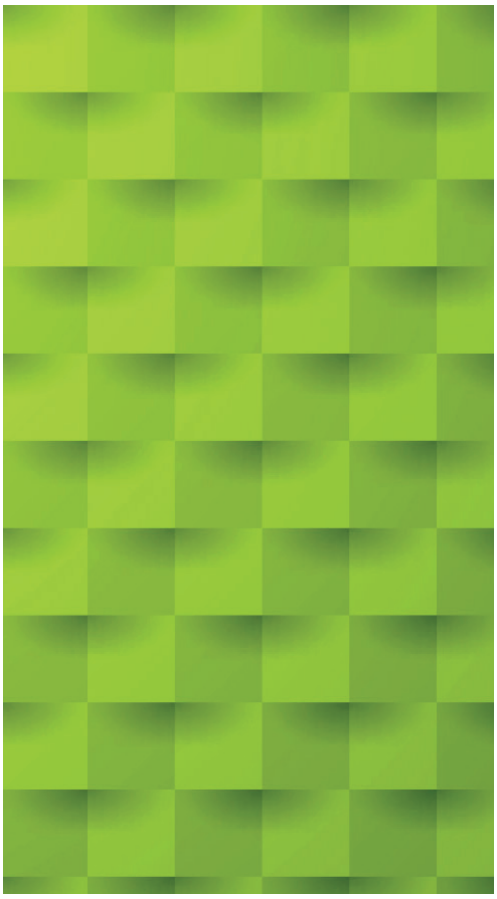


Una arquitectura más verde

Independientemente de lo sencillo o complejo que sea el software, de lo estrictos o simples que sean los requerimientos, es esencial desarrollar inicialmente una visión del resultado esperado. Esto puede ir desde una simple frase conceptual hasta un detallado documento de arquitectura. Sea cual sea el enfoque adoptado, será la base de todas las decisiones posteriores.

En consecuencia, cuestionar el plan tendrá probablemente importantes implicaciones en términos de costes y plazos (lo que también implica que se puede desperdiciar energía).

A continuación, exploraremos algunos de los principios generales que ayudarán a definir una visión más completa y sostenible del desarrollo sostenible.



Apagar cuando no se utilice r

Los principios básicos del ahorro de energía en casa pueden (y deben) aplicarse también al diseño de software. De la misma manera que hay que apagar las luces cuando no hay nadie en una habitación, hay que apagar el software cuando nadie lo utiliza. Siguiendo con esta analogía, hay que tener en cuenta que nos limitamos a apagar las luces en lugar de desenchufar o desactivar la energía de toda la casa. En consecuencia, las aplicaciones deben diseñarse basándose en principios modulares para que puedan apagarse por separado.

Este enfoque es fundamental para los microservicios y arquitecturas sin servidor, no sólo cuando se trata de apagados totales, sino también con respecto a la escalabilidad o a la puesta en marcha y parada de módulos de réplica en todo el mundo debido a las fluctuaciones de la demanda. Las decisiones basadas en este enfoque se verán reflejadas en la implementación y el despliegue finales (que se trata también más adelante en este documento).

Lograr el equilibrio adecuado puede ser un arte, especialmente en ausencia de experiencias previas con las que comparar y con las que poder establecer una línea base. Si ese es el caso y se trata de un producto nuevo, se pone en marcha de todos modos y se hace un seguimiento de los patrones de uso reales para ver cómo se comportan los sistemas e identificar posibles optimizaciones. Independientemente de lo que se sepa sobre los niveles de demanda, el código debería estar siempre en condiciones de separar secciones individuales y transformarlas en módulos nuevos y autónomos. Esto ya se ha destacado como la mejor práctica a la hora de escribir el código para este tipo de diseños, pero también es útil prever cómo se quieren dividir las secciones individuales por separado, para no mantener los componentes acoplados de forma involuntaria.

Evitar el consumo impulsivo

Profundizando y analizando los servicios internos, otra forma de mejorar la eficiencia puede ser priorizar la resiliencia orientando ciertas secciones del software hacia una posible falta de disponibilidad.

Un enfoque común es añadir asincronía de procedimientos; “usar mal” deliberadamente los sistemas y agrupar los trabajos para procesarlos juntos y en secuencia. Si no se comprueba si las tareas requieren un procesamiento en tiempo real, todo el procesamiento se realizará automáticamente en el modo de tiempo real por defecto. Si se comprueba que no es necesario el procesamiento en tiempo real, se puede llegar a la conclusión de que el procesamiento al final del día podría ser el enfoque más válido y eficiente a adoptar. Incluso el procesamiento una vez por hora podría marcar la diferencia, especialmente si los requisitos permiten procesar los volúmenes en lotes consolidados en un momento posterior.

También se podría aplicar este concepto a los entregables para mejorar la eficiencia general. Algunas operaciones que se completan en tiempo de ejecución también podrían producirse durante el tiempo de construcción. Un ejemplo es la introspección de código (que permite que ciertas secciones de código inspeccionen otras y/o generen nuevo código en tiempo de ejecución), asegurando que no haya posibilidad de realizar la generación de código durante la compilación cuando se está generando el archivo final ejecutable. Otro ejemplo podría ser la página de inicio de una aplicación web; donde las secciones interiores pueden estar basadas en contenido realmente dinámico, pero también el contenido inicial de la página puede cambiar cada día. En esta situación podemos considerar el uso de técnicas de generación de sitios estáticos, reconstruyendo la página una vez (por lanzamiento, por semana, por día o incluso por hora, dependiendo de sus necesidades) y transformando cualquier contenido dinámico en contenido estático que es mucho más fácil de optimizar en términos de consumo de energía. Estos ejemplos (y muchos otros escenarios) también podrían resolverse utilizando el almacenamiento en caché en la línea de los procedimientos “lazy build”. No hay nada de malo en resolver los problemas de esta manera, siempre y cuando sigamos desafiando los requerimientos del “justo a tiempo”.

Focalizar la inversión de tiempo y energía

Al integrar los criterios de GreenCoding en un diseño de arquitectura, siempre hay que tener en cuenta el ciclo de vida del software: su creación, uso, mantenimiento y eliminación. Para tener una visión global, siempre hay que tener en cuenta a quién que va dirigido el software. La primera consideración será examinar los factores relacionados con los humanos o las máquinas; ¿lo utilizarán usuarios humanos u otros sistemas? ¿Cuántos usuarios esperamos? ¿Docenas o tal vez miles? Estas líneas de investigación deben continuar para asegurar que se entienden correctamente las frecuencias de uso esperadas y la duración de las interacciones medias del software.

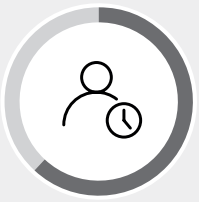
Este análisis debería proporcionar una buena comprensión de los elementos de la arquitectura que requerirán más energía. Por ejemplo, en el caso de un software de back-office alojado en un servidor compartido, no es raro que cualquier desarrollo genere una huella energética importante si el software sólo se utiliza realmente durante un par de minutos a la semana (si es que se utiliza). Esto contrasta, por ejemplo, con una simple aplicación de horarios utilizada por una universidad, que puede tener que generar miles de imágenes cada hora. Ahorrar incluso un 0,1 segundo en la creación de una imagen de este tipo podría suponer un importante volumen de energía al año.

Un buen ejemplo de cómo incluso un pequeño ahorro de tiempo puede tener un gran impacto podría ser la optimización del tiempo de inicio de una aplicación bancaria virtual utilizada por 500.000 clientes. Sustituyendo las imágenes de la pantalla de carga y reduciendo su resolución al nivel adecuado, los tiempos de apertura pueden reducirse, incluso en un milisegundo. Suponiendo que el usuario medio abra la aplicación al menos una vez al día, esto podría ahorrar 50 horas (¡o 2 días!) de tiempo de funcionamiento en los dispositivos móviles al año.

Las cifras que comentamos aquí deberían ser ya fundamentales en el proceso de comprensión de los requisitos no funcionales y en la estimación de los esfuerzos iniciales y la inversión de tiempo en cualquier proyecto. Desde la perspectiva de GreenCoding, tiene sentido llegar a comparaciones reales de la inversión de tiempo para cada fase del desarrollo. Esto puede ser tan simple o tan complejo como sea necesario, pero en aras de la claridad, veremos un caso de uso relativamente sencillo. Si examinamos la huella energética de un jefe de equipo, un desarrollador y un tester que utilicen un ordenador portátil, podemos dotar a cada uno de ellos de 5 “unidades”. También podemos dotar al servidor de 20 unidades, mientras que cada usuario de un dispositivo móvil supondrá 1 unidad. Este planteamiento inicial puede ser demasiado simplificado, pero incluso un modelo tan sencillo te dará una buena idea general. Si quieres hacerlo más granular, puedes añadir fácilmente tantas capas como quieras.

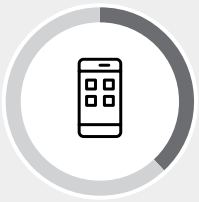
DESARROLLO

Total de horas de trabajo (humanas) necesarias para que la aplicación esté disponible



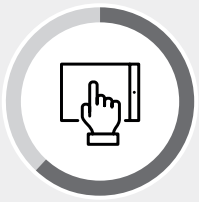
MANTENIMIENTO PROYECTADO

Total de horas de trabajo al año



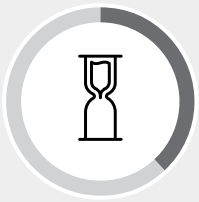
TIEMPO DE EJECUCIÓN

Tiempo de funcionamiento del servidor al año durante la vida útil prevista



TIEMPO DE USO

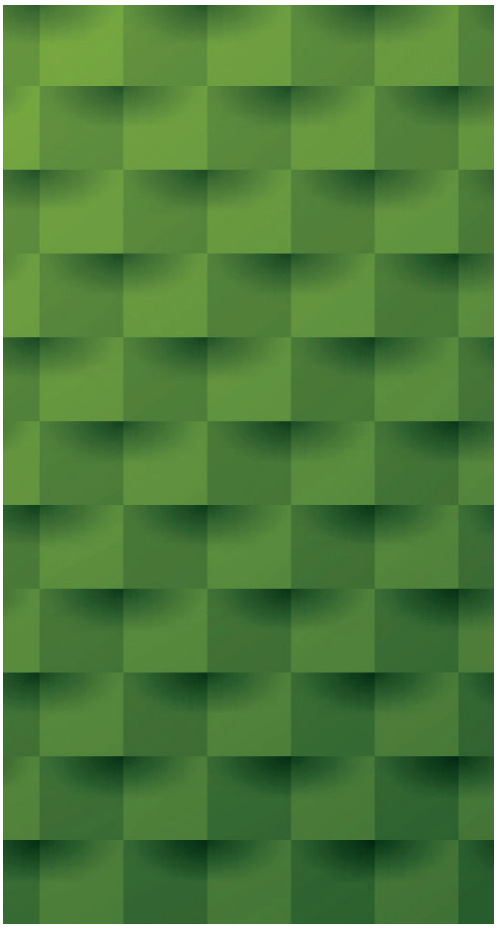
Estimación del tiempo que se tarda en completar una sola interacción del usuario (expresado como un caso de uso estándar / journey del cliente, que se multiplica por el número previsto de interacciones del usuario al año durante la vida útil prevista)

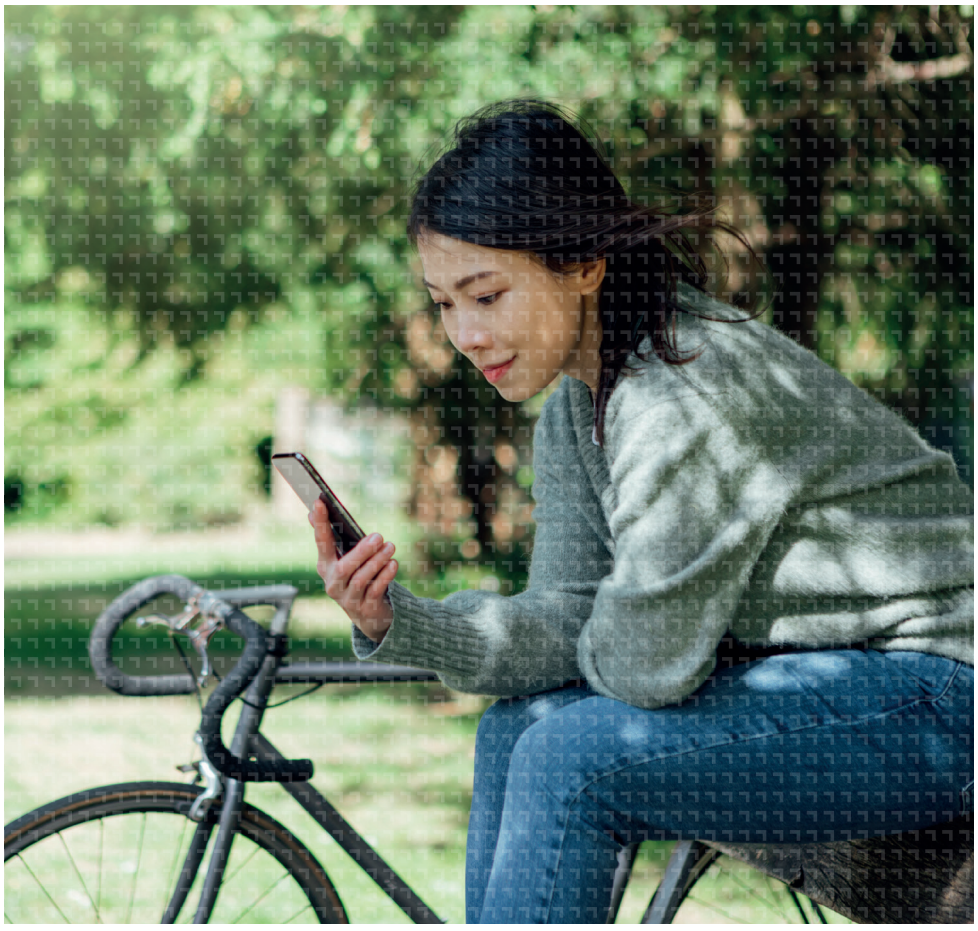


TIEMPO DE ESPERA

Estimación del tiempo empleado en la pantalla “Cargando”... para una sola interacción del usuario (expresado como un caso de uso estándar / journey del cliente, que luego se multiplica por el número previsto de interacciones por año a lo largo del tiempo de vida previsto)

Disponer de una visualización del tiempo invertido en cada etapa del ciclo de vida del software -aunque sólo ofrezca una idea general- ayudará a toda la cadena de desarrollo de software a comprender mejor dónde centrar los esfuerzos para reducir la huella energética.





Adoptar una lógica más verde

Una vez establecidas las bases de la arquitectura, es hora de hacer las cosas tangibles. A menudo parecerá que las decisiones de desarrollo de cada día tienen poco impacto en el rendimiento general, pero en realidad no es así y hay que evitar caer en esta trampa. Cada decisión es importante. Es raro tener la suerte de reducir el tiempo de carga en un 20% sólo por llevar a cabo dos horas de recodificación, sin embargo, a veces esto puede ser posible. No obstante, estos esfuerzos son necesarios si se quiere alcanzar el objetivo de implementar la sostenibilidad en el diseño del software. Asimismo, aunque el impacto de una determinada decisión pueda parecer insignificante de forma aislada, nunca deben ignorarse las ramificaciones de una decisión individual, ya que éstas se multiplican. Diez impactos “prácticamente insignificantes” pueden combinarse para marcar una diferencia notable. El rendimiento siempre es importante, independientemente de la escala.

El objetivo de esta sección en este documento es proporcionar al lector algunas ideas sobre cómo determinar las posibles mejoras de rendimiento de GreenCoding. Algunas de ellas pueden aplicarse a un proyecto en curso o a un futuro desarrollo previsto. Algunas pueden suponer un gran beneficio con determinados tipos de software. Con otros, el tiempo y el esfuerzo invertidos no merecerán la pena por el ahorro obtenido. Lo importante es que todo el mundo sea consciente de estas cuestiones y que sean un punto de referencia para futuras mejoras.

Código con cero residuos

Desde una perspectiva técnica y de desarrollo, la codificación consiste fundamentalmente en resolver problemas. Al principio, las empresas desarrollaban soluciones a medida para cada problema y, aunque no tardaron en empezar a ofrecer software de acceso público, la verdadera revolución llegó con el software de código abierto y la distribución gratuita de licencias. Se pusieron a disposición del público piezas de código listos para resolver problemas grandes o pequeños, código que podía seguir combinándose y ampliándose.

Como resultado, los plazos de entrega y las inversiones en recursos se han reducido considerablemente con el tiempo, lo que es, por supuesto, un avance positivo. Sin embargo, al igual que en el mundo físico del ecologismo, el hecho de que algo sea gratuito no significa que deba utilizarse de forma irresponsable. Hasta el 90% del software moderno contiene código abierto desarrollado por terceros y, aunque esto no es necesariamente malo, potencialmente el problema a resolver puede ser un ajuste exacto con un diseño de biblioteca preexistente. Además, a medida que se dispone de más recursos externos, aumenta la posibilidad de que se introduzcan secciones de código redundantes (o que ya lo hayan hecho).

Esto es especialmente importante en el caso de las aplicaciones web que requieren la descarga e instalación de software cada vez que un usuario las visita. Aunque los desarrolladores puedan "consumir el código de forma gratuita" y tenga un impacto potencialmente insignificante en los tiempos de compilación, si el código se utiliza pero no es realmente necesario para que los usuarios obtengan los beneficios esperados, puede desperdiciar tiempo de red y de análisis de la CPU. Hay que recordar que los navegadores analizarán cualquier información que se les proporcione para poder ejecutarla cuando sea necesario, por lo que es responsabilidad de los desarrolladores web garantizar que las cargas de trabajo sean eficientes.

Para las aplicaciones móviles o de escritorio, el tiempo de red es menos relevante si las aplicaciones sólo se descargan una vez por cada usuario. Sin embargo, el tiempo de arranque seguirá viéndose afectado por el tamaño del paquete de software.

El software que se ejecuta siempre en servidores dedicados depende mucho menos del tamaño del paquete, porque sólo se instala una vez al trimestre, por ejemplo. El volumen de datos que se carga para iniciar el software tiene menos impacto en el procesamiento general, pero si el requisito es pasar a infraestructuras cloud o utilizar arquitecturas de microservicios, es importante considerar cuánto esfuerzo se requiere para mover la aplicación de un servidor a otro. Los cloud y los clústeres hacen que el consumo de energía sea más contenido, pero, una vez más, esto no significa que se deba subestimar su huella de carbono.

Los tiempos de arranque siguen siendo relevantes y esto puede afectar a toda la estrategia; se puede considerar apropiado cerrar una API no crítica porque su tiempo de arranque es de un segundo y no supera el retraso asumido para el operador de aproximadamente un segundo. Si el retraso fuera mucho mayor, por ejemplo cinco segundos, podría no ser aceptable. En este caso, la API no debería cerrarse nunca o debería escalarse de forma más adecuada.

Una vez más, todo pueden abordarse desde un ángulo **preventivo** o **correctivo**.

Los problemas de tamaño de los paquetes pueden evitarse asignando un presupuesto de tamaño (o un presupuesto de rendimiento). Esto implica definir el tamaño que deben tener las aplicaciones e introducir comprobaciones automáticas para avisar a los desarrolladores en el momento en que se agote el presupuesto. Los presupuestos también pueden utilizarse para evitar que se realicen grandes

Si un proyecto ya está en marcha, siempre existen formas de introducir medidas correctoras para reducir el tamaño del software.

El primer paso es centrarse en el código que nunca se ejecutará. Como se ha mencionado anteriormente, hay que prestar atención a las líneas de código que no han sido escritas por el equipo, pero que se refieren a una dependencia para que el código se ejecute. Hacer esto manualmente puede llevar mucho tiempo y ser extremadamente arriesgado. Por lo tanto, una buena forma de localizar y eliminar el código "muerto" de forma segura es utilizar motores de "tree-shaking" (agitar el árbol es la metáfora de esta técnica, el código que no se use, que no esté "bien conectado" acaba cayendo y siendo eliminado).

El impacto del tamaño del paquete no es el mismo en todos los ámbitos; por ejemplo, es más común que estos motores se apliquen a los lenguajes informáticos utilizados para el desarrollo web (como Javascript, donde todos los principales paquetes incluyen esta función por defecto). También es posible encontrar herramientas de este tipo utilizadas para lenguajes backend típicos (por ejemplo, ProGuard para Java y Kotlin). No obstante, se trata de una técnica muy delicada que requiere un enfoque adecuado con respecto a la biblioteca (que proporciona una modularización de grano fino), el desarrollador (que tendrá que hacer referencias precisas a los módulos de la biblioteca) y el propio motor de "tree-shaking" (que combina toda la información basándose en un enfoque inteligente).

Los motores de "tree-shaking" son muy sensibles, por lo que todo depende del desarrollador y de su atención a los detalles. Introducir una dependencia de terceros en un proyecto tiene que pensarse cuidadosamente. Es similar a invitar a alguien como huésped a tu casa. Como mínimo, el desarrollo de software sostenible requiere una idea clara del impacto que cada "invitado" puede tener en el resultado final. Por supuesto, los desarrolladores web tendrán una mejor visión del código entregado, por ejemplo, dispondrán de herramientas más avanzadas como los visualizadores de paquetes, que proporcionan un mapa visual (cajas o gráficos circulares) que muestran el tamaño relativo de cada sección del código (incluido el código tomado de fuentes abiertas). Otros lenguajes de codificación pueden no proporcionar este tipo de enfoque, lo que requiere una codificación más manual. Esto también permite modificar las bibliotecas de terceros en lugar de limitarse a adoptar el contenido de la biblioteca sin realizar ajustes. En última instancia, las técnicas de la "vieja escuela", como el acceso a la carpeta de la biblioteca y el análisis del tamaño de los archivos, también pueden funcionar bien.

Una vez comprendido el impacto de cada dependencia en el software, se puede priorizar dónde mejorar la eficiencia. A veces, la relación beneficio-impacto es escasa, por lo que habrá que sopesar la probabilidad de éxito de sustituir la biblioteca pesada por una alternativa (si no se puede adaptar). Por supuesto, la biblioteca en sí misma puede estar absolutamente bien tal y como está diseñada, pero puede ser que simplemente no se adapte a las necesidades del sistema (por ejemplo, tal vez utilizar una enorme biblioteca de gráficos para dibujar un único gráfico de barras que podría crearse manualmente). Llegados a este punto, merece la pena mencionar el uso sostenible del software de código abierto, ya que esto está impulsando actualmente un mayor enfoque entre los proveedores de código abierto sobre el impacto de sus librererías, especialmente en lo que respecta a las capacidades de “tree-shaking”. Del mismo modo que los supermercados empezaron a introducir productos respetuosos con el medio ambiente (y la concienciación de los consumidores dio forma a toda la cadena de suministro), el GreenCoding también tiene el potencial de crear un cambio de paradigma que podría dar forma a todo el área de las soluciones de código abierto.

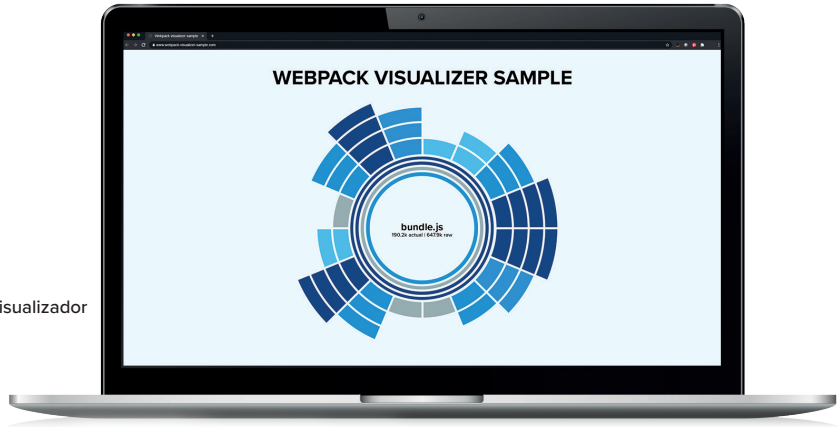


Imagen:
Ejemplo de visualizador de paquetes

Recursos de baja intensidad

Aunque los desarrolladores deberían centrarse inicialmente en la intensidad energética del propio código, siempre existe la posibilidad de realizar más optimizaciones si se tienen en cuenta otros recursos que requiere el software [9].

Por ejemplo, el modo en que se organiza la información puede influir en la eficiencia del software, dependiendo de la cantidad de recursos que se utilicen. Puede que no cueste mucho esfuerzo analizar un archivo estructurado de forma ineficaz una vez a la semana, pero si hay que transmitirlo por una red cientos de veces cada hora, está claro que no es lo ideal.

De nuevo, la concienciación suele ser lo más difícil. La opción obvia es pensar en utilizar diferentes formatos de archivo, sustituyendo quizás una hoja de cálculo Excel por un simple archivo CSV. O un archivo XML puede almacenarse como un archivo YAML. De todos los recursos de la aplicación, hay uno que probablemente destaque más cuando se trata del impacto global en el uso de la energía: las imágenes.

Una de las cuestiones que más se pasan por alto en relación con las imágenes es cómo se empaquetan. La primera decisión que hay que tomar es la de utilizar imágenes de trama (como los mapas de bits) o imágenes vectoriales basadas en líneas y formas simples. Una buena regla general es utilizar imágenes rasterizadas para fotos o dibujos detallados y utilizar imágenes vectoriales para logotipos, símbolos y gráficos. Las imágenes rasterizadas deben tener un tamaño adecuado. Utilizar una imagen detallada de 10 MB para una referencia en miniatura es un enorme gasto de recursos. El procesamiento de imágenes ha avanzado a pasos agigantados en los últimos años y los formatos de imagen eficientes de nueva generación (como webp) se han diseñado específicamente para la transmisión en red.

Las imágenes vectoriales se diseñaron pensando en la escalabilidad, así que, aunque esto suena bien, no existe un tamaño por defecto. En cuanto al formato, SVG es el estándar de facto para las imágenes vectoriales, y difícilmente puede mejorarse. Dicho esto, siempre aconsejamos a los desarrolladores que optimicen las redes y el procesamiento agrupando las imágenes vectoriales en archivos únicos (utilizando sprites).

Siguiendo con el tema de los contenidos visuales, pero pensando en las cosas desde un ángulo diferente, las impresiones visuales también pueden tener un impacto en el consumo de energía. El concepto emergente de “dark design” o diseño oscura no sólo está cambiando las preferencias de diseño, sino que también está revelando nuevas formas de reducir la cantidad de energía que utilizan las pantallas. En combinación con la tecnología de pantallas OLED, que se utiliza principalmente en los smartphones, el modo oscuro puede reducir el uso de la batería hasta en un 23,5% [10]. En consecuencia, ofrecer un “dark mode” o modo oscuro debería ser una prioridad, especialmente si se desarrolla una aplicación web o móvil, aunque esto también tiene implicaciones para el marketing y el diseño de la marca. Ofrecer a los usuarios la opción de esquemas de color alternativos (más oscuros) y más eficientes desde el punto de vista energético puede reforzar la marca al añadir funciones que el usuario activa conscientemente. Naturalmente, algunos usuarios estarán más inclinados a responder a estas opciones que otros, pero se puede fomentar la adopción sugiriendo el cambio al “dark mode” en determinadas condiciones de iluminación.

[9] “The cost of JavaScript in 2019”: https://v8.dev/blog/cost-of-javascript-2019, June 2019

[10] “What is the impact of Dark Mode on battery drain”: http://mobileenerlytics.com/dark-mode/, August 2019

Feeds de proximidad

Aunque los programas informáticos están concebidos para ofrecer ventajas al usuario, tratar adecuadamente las fechas de caducidad de la información puede tener un gran impacto en los recursos necesarios. Cualquier información proporcionada al usuario debe ser clasificada en función de su duración requerida; la información puede ser válida durante segundos, a veces se necesita durante días, semanas o incluso por un periodo de tiempo ilimitado. La combinación de este aspecto con la frecuencia con la que se solicita la información aclara las necesidades de almacenamiento en caché y las posibles ventajas de eficiencia.

El almacenamiento en caché (guardar los datos relevantes temporalmente en una capa intermedia) puede realizarse en varios puntos del ciclo de vida de la información:

Cuanto más cerca esté la caché del usuario, mejor. A veces es útil aplicar el almacenamiento en caché a varias capas.

Cabe mencionar aquí el impacto que las redes sociales y el almacenamiento distribuido han tenido en la actualización que los usuarios esperan de la información. El aumento de la eficiencia puede repercutir en la experiencia del usuario, por lo que, por ejemplo, puede merecer la pena revisar el enfoque de las imágenes que se entregan a los usuarios. ¿Qué buscan los usuarios? ¿Necesitan una vista completa o sólo buscan algunos resultados? ¿Cuál será el impacto de retener algunas imágenes durante uno o dos minutos? ¿O esperar una hora? ¿O incluso un día?

Es sorprendente cómo incluso los compromisos más sencillos pueden mejorar significativamente la eficiencia y cómo a menudo es totalmente aceptable hacer tales actualizaciones.

Sin embargo, en términos de aplicación práctica, es aconsejable que un arquitecto de datos o de información analice todos los flujos de datos durante la fase de diseño. Dicho esto, las aplicaciones modernas se basan en arquitecturas estratificadas o distribuidas, que deberían proporcionar métricas detalladas de uso continuo. El análisis de las métricas y la identificación de posibles optimizaciones de los feeds en el entorno real (mediante la localización de las solicitudes y los orígenes más frecuentes) sería la mejor manera de introducir mejoras, basadas en el comportamiento real de los usuarios.

Esto es especialmente importante en el caso de las aplicaciones web, ya que estos sistemas transmiten su propio software de interfaz en todas y cada una de las visitas de los usuarios. También hay un mayor margen de optimización potencial en este ámbito, ya que es fácil identificar el contenido “casi permanente” (por ejemplo, un logotipo). Como resultado, surgen un par de conceptos, además del almacenamiento en caché HTTP habitual:

01.
Aplicaciones web progresivas (PWA, según siglas en inglés: los navegadores modernos pueden transformar las páginas web compatibles con los estándares PWA en aplicaciones. Esta técnica proporciona capacidades lógicas más elaboradas en cuanto a la gestión de la caducidad del contenido, así como soporte offline [11,12].

02.
Redes de distribución de contenidos (CDN, según siglas en inglés): en ubicaciones periféricas: una CDN es una plataforma altamente distribuida de servidores que ayuda a minimizar los retrasos en la carga del contenido de las páginas web al reducir la distancia física entre el servidor y el usuario. Los proveedores de CDN han creado soluciones inteligentes y adaptables para facilitar la carga de aplicaciones web y, en algunos casos, también pueden actuar como una capa adicional de almacenamiento en caché de datos entre los servicios y la interfaz visual.

[11] “Impact of Progressive Web Apps on Web App Development”: https://www.researchgate.net/publication/330834334_Impact_of_Progressive_Web_Apps_on_Web_App_Development, September 2018

[12] “Progressive Web Apps for the Unified Development of Mobile Applications”: https://link.springer.com/chapter/10.1007/978-3-319-93527-0_4, June 2018

Métodos más verdes

Feedback rápido, mejores decisiones

Uno de los principales retos para tener éxito con GreenCoding es la concienciación. Todas las técnicas descritas anteriormente son respuestas naturales para aumentar la eficiencia una vez que las partes interesadas son conscientes de la cantidad de energía que se desperdicia. Por tanto, cuanto antes sea un equipo consciente del impacto energético de sus decisiones, más fácil será mejorar la toma de decisiones. Esto apunta directamente a los métodos de feedback, que deben organizarse en ciclos rápidos.

A nivel general, los métodos agile y lean ofrecen mejores oportunidades para adaptar el software a los principios de ofrecer pantallas orientadas al beneficio, como se ha descrito en la sección anterior [13]. A un nivel más profundo, aplicar la integración continua y la entrega continua permite visualizar el impacto final de cada decisión de desarrollo [14].

Una vez que un equipo tiene implantados los ciclos de feedback rápido, tendrá que definir las métricas que se utilizarán, basándose en las suposiciones relativas a GreenCoding. Por supuesto, hay un montón de métricas complejas a las que se podría hacer seguimiento, pero el mejor retorno de la inversión es más probable que con los tiempos de carga. Estos son fáciles de medir (y de observar a simple vista) y se correlacionan directamente con el consumo de energía.

Los equipos deben mantener registros de los tiempos de carga iniciales y de las principales interacciones. La obtención de estas métricas suele ser una práctica habitual en los proyectos, pero sólo en las fases finales para preguntarse si es funcionalmente aceptable.

Esta mentalidad debe cambiar para mejorar la eficiencia: aunque los tiempos de carga puedan ser aceptables desde el punto de vista de la usabilidad, si una aplicación se analizara según los principios de GreenCoding, los tiempos de carga lentos no se considerarían sostenibles.

Hacer un seguimiento de los tiempos de carga e interacción desde el principio es una forma muy eficaz de centrarse en la sostenibilidad.

En la práctica, el equipo de desarrollo debería empezar por lo básico (es decir, los tiempos de carga de “Hello world”) y luego etiquetar cada nueva función y su impacto. Obviamente, las funciones iniciales, como el acceso al almacenamiento, tendrán un impacto significativo. Es esencial asegurarse de que existe una imagen clara del antes y el después para que, sea cual sea el impacto, haya suficiente información para evaluar adecuadamente si el valor aportado por el cambio ha supuesto un beneficio que merezca la pena. Siempre que se cuestionen los incrementos significativos, se cumplirán los objetivos de sostenibilidad. En raras ocasiones, puede haber rutinas obvias que consumen muchos recursos y que desempeñan un papel definitorio en los tiempos de carga o interacción que ocultan involuntariamente ineficiencias subyacentes. Para revelar dichas ineficiencias, se deben crear escenarios en los que los procesos derrochadores se sustituyan por respuestas “pregrabadas” casi instantáneas, proporcionando un conjunto secundario de métricas de rendimiento que no se ven afectadas por las rutinas que consumen muchos recursos.

Aunque se anima a los desarrolladores a aplicar técnicas de integración continua (CI, según siglas en inglés) y de entrega continua (CD, según siglas en inglés) para apoyar los ciclos rápidos de feedback, también debe prestarse atención a la configuración. Las configuraciones descuidadas de las herramientas de CICD pueden llevar a que las complejas pruebas de integración se ejecuten automáticamente con más frecuencia de la deseada o incluso necesaria. Es esencial garantizar que los equipos de desarrollo alcancen el equilibrio adecuado entre la obtención de feedback automático y el mantenimiento de recursos suficientes para procesar o reaccionar al feedback. Una forma de conseguir un gran ahorro, tanto en términos de consumo de energía directo (tiempo de CPU) como indirecto (tiempo que pasan los desarrolladores esperando la retroalimentación), es aplicar técnicas de compilación y testing incremental [15]. Esto implica recompilar sólo secciones modificadas de código en lugar de entregas enteras, o sólo probar el código reescrito y los sistemas afectados por el código recompilado. Los resultados de la compilación también deberían compartirse entre los desarrolladores y los sistemas CICD.

[13] “Lean and Agile: differences and similarities”: https://twproject.com/blog/lean-agile-differences-similarities/, November 2018

[14] “Continuous integration vs. continuous delivery vs. continuous deployment”: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

[15] “Incremental Model in SDLC: Use, Advantage & Disadvantage”: https://www.guru99.com/what-is-incremental-model-in-sdlc-advantages-disadvantages.html, November 2020

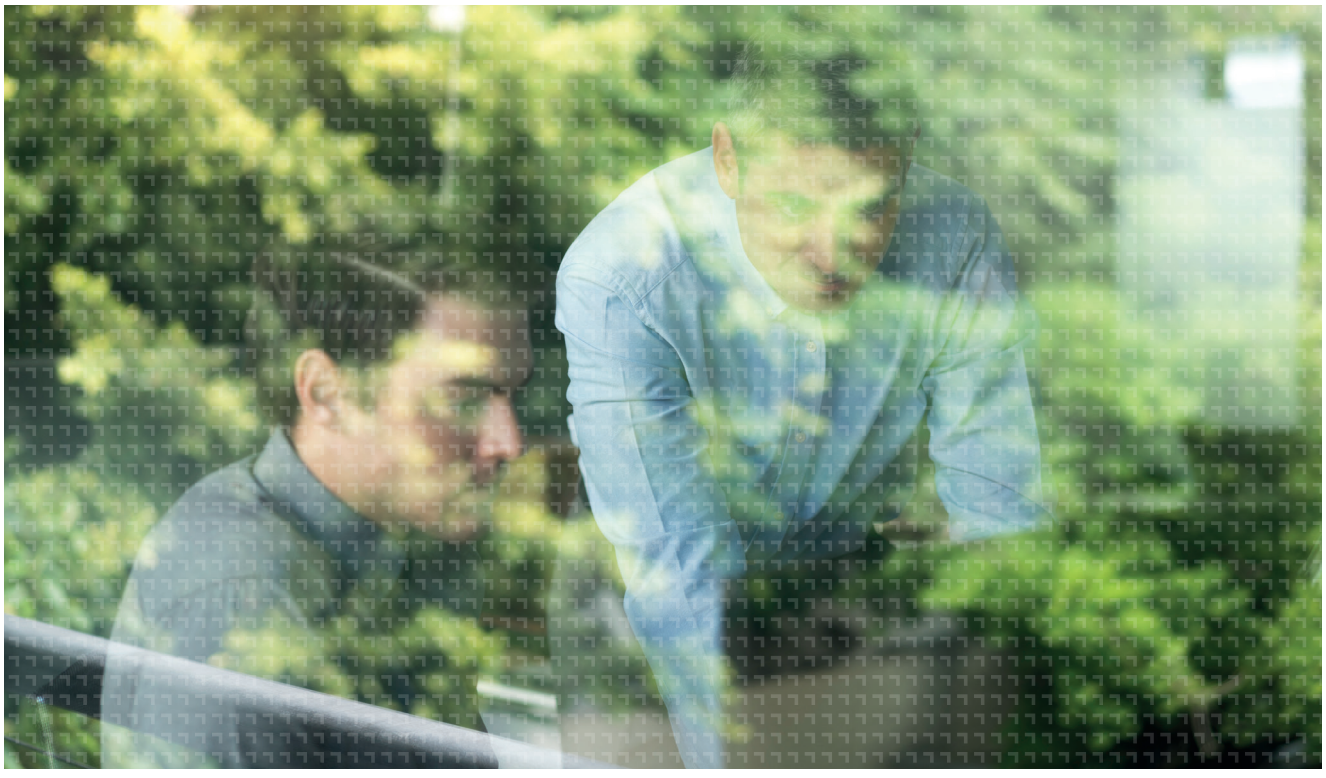
Resultados reutilizables

Comprender la oportunidad de hacer que el software sea más sostenible requiere tiempo y esfuerzo, pero es un punto de partida importante. Los equipos, las empresas y las unidades de negocio son heterogéneos por naturaleza y, en consecuencia, también adoptarán naturalmente diferentes enfoques de GreenCoding. Es muy posible que las organizaciones necesiten introducir y examinar datos de referencia durante varios días. En otros casos, es posible que sólo se necesiten un par de minutos y unos simples bocetos para entender lo que está ocurriendo. En cualquier caso, en la mayoría de los casos el equipo habrá empezado en una posición de incertidumbre, enfrentándose a retos y nuevos requerimientos, lo que implica invertir tiempo y energía en generar nuevos resultados. La mejor práctica es organizar y compartir todo esto de forma que sea accesible y legible para los demás miembros del equipo, para toda la organización o incluso para la comunidad TI en general.

Parte del desarrollo de software sostenible es garantizar que los resultados de los proyectos de GreenCoding estén disponibles para los demás y puedan ser buscados por los miembros del equipo, las personas de la organización o incluso la comunidad en general, y este procedimiento también debe seguir siendo eficiente en sí mismo.

Dejar que la comunidad de desarrolladores vuelva a abordar las mismas cuestiones y problemas por sí misma significa tener que empezar desde cero (incluso en un proyecto de GreenCoding), lo que claramente supone una pérdida de tiempo, esfuerzo y, por tanto, energía.

Es importante destacar las repercusiones positivas de introducir cambios en la eficiencia de los proyectos por varias razones. Al establecer una métrica sobre las contribuciones a la eficiencia realizadas al emprender acciones específicas dentro de los proyectos, estos beneficios pueden extrapolarse a otros proyectos con condiciones similares. Disponer de estos parámetros evita la necesidad de cuantificar los beneficios por segunda vez mediante la realización de evaluaciones comparativas, ya que las nuevas acciones se basan en las ventajas asumidas durante el proceso de diseño. En última instancia, el objetivo de cualquier equipo que aplique los principios de la codificación verde debe ser minimizar el número de medidas correctivas necesarias, recurriendo a un arsenal fiable de medidas preventivas. Como ya se ha señalado, los contextos de los proyectos son heterogéneos por naturaleza, por lo que nada en este ámbito es meramente “blanco o negro”; el éxito depende, en última instancia, de la capacidad de captar e indexar conjuntos personalizados de medidas preventivas que sean pertinentes para los proyectos de desarrollo.



Una plataforma más verde

Si consideramos GreenCoding desde una perspectiva más amplia, las cuestiones relativas a la infraestructura adecuada para ejecutar el código son tan cruciales como el propio código. Cuando se trata de hardware y potencia informática, los niveles de utilización son cruciales. ¿Por qué los niveles de utilización influyen tanto en la eficiencia energética?

Utilización óptima

La energía consumida por un sistema informático no es proporcional a los niveles de utilización.

Este concepto se conoce como proporcionalidad energética, ya que a mayores niveles de utilización se produce un menor consumo de energía por punto porcentual de potencia de cálculo utilizada.

Los bajos índices de utilización de los servidores son un problema común. Suelen deberse a la tendencia a sobrestimar durante la planificación la cantidad de software y, por tanto, la capacidad del servidor que se utilizará realmente. Por ejemplo, los desarrolladores pueden prever un número excesivo de usuarios o esperar que se incorporen más usuarios más adelante. Basándose en sus suposiciones incorrectas, la potencia informática se extrapola a menudo a lo largo de varios años, lo que da lugar a sistemas sobredimensionados. Dado que la mayoría de los sistemas ejecutan múltiples aplicaciones, puede ser imposible precisar el consumo de energía de una aplicación específica, como un programa que se ejecuta en un monolito compartido, junto con otras aplicaciones.

Una forma eficaz de averiguar lo que está ocurriendo y de hacer un seguimiento del consumo de energía de una aplicación específica es utilizar el cloud, no porque los proveedores cloud sean mejores en matemáticas y tengan números más precisos, sino simplemente porque en un sentido operativo se gastan menos recursos. Con el cloud computing, se ven correlaciones directas: cuanto más alta es la factura, más energía se ha consumido.

Más dinero gastado en cloud computing suele ser igual a más energía consumida.

El uso del cloud también puede influir significativamente en el consumo de energía. Como se ha descrito anteriormente, los niveles de utilización de hardware más altos también son más eficientes desde el punto de vista energético. Por ello, el cloud computing

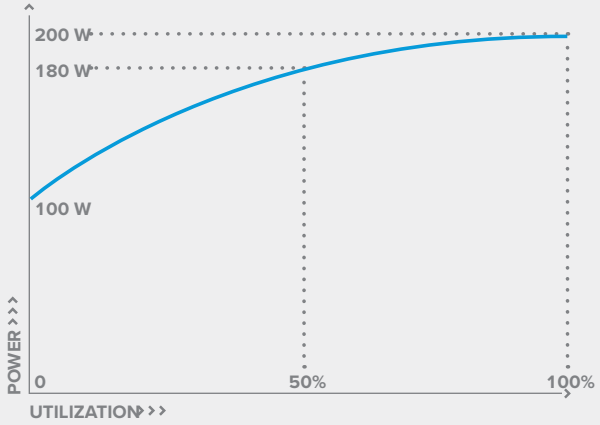


Figura 3: Proporcionalidad energética

ofrece un potencial útil para ahorrar energía. Incluso cuando los servidores internos/locales están inactivos, siguen consumiendo energía. Los sistemas cloud públicos se basan en principios de alta modularidad, lo que permite controlar los niveles de utilización con mayor precisión que con los sistemas no modulares, especialmente si se utilizan módulos que no pueden desconectarse para que dejen de consumir energía cuando no están en uso.

Según un documento publicado por el Consejo de Defensa de los Recursos Naturales (Natural Resources Defense Council, NRDC), los niveles de utilización de los servidores cloud de grandes proveedores como AWS, Google Cloud y Microsoft Azure se sitúan en torno al 65%[16]. Esto se compara con los centros de datos locales, que suelen funcionar con niveles de utilización de entre el 12% y el 18%. Teniendo en cuenta el concepto de proporcionalidad energética mencionado anteriormente, esto conlleva desventajas significativas en términos de eficiencia energética.

Otro aspecto importante a la hora de mejorar la eficiencia energética es el diseño de la infraestructura técnica, por ejemplo, la tecnología de refrigeración. Las grandes inversiones en la eficiencia de los sistemas se transformarán en importantes reducciones de costes para los proveedores, lo que ofrece una gran motivación para aunar las ganancias medioambientales con los beneficios económicos. Las mejoras en la infraestructura a gran escala pueden ayudar a reducir el consumo de energía hasta un 29% en comparación con los típicos centros de datos locales [17] Google Cloud ha llevado la eficiencia un paso más allá y ahora utiliza el aprendizaje automático para reducir la energía necesaria para la refrigeración hasta en un 40% [18].

La última parte de la ecuación cuando se trata de minimizar la huella de carbono del cloud computing es la fuente de energía, o el llamado mix energético . Un objetivo que todos los proveedores de cloud pública tienen en común es confiar únicamente en la energía renovable para alimentar la infraestructura cloud. Algunos están más cerca de lograr este objetivo que otros.

Los altos niveles de utilización, el diseño eficiente de la infraestructura y una combinación de energía limpia son, por tanto, factores clave a la hora de minimizar el impacto medioambiental del cloud computing. Son áreas en las que los grandes proveedores cloud tienen claras ventajas. Incluso si la tecnología que ofrecen es intensiva en energía, están en condiciones de mejorar sistemáticamente.

[16] NRDC, "Data Center Efficiency Assessment": https://www.nrdc.org/sites/default/files/data-center-efficiency-assessment-IP.pdf, August 2014

[17] AWS News Blog, "Cloud Computing, Server Utilization, & the Environment": https://aws.amazon.com/blogs/aws/cloud-computing-server-utilization-the-environment/, June 2015

[18] Google Blog, "DeepMind AI reduces energy used for cooling Google data centers by 40%": https://blog.google/outreach-initiatives/environment/deepmind-ai-reduces-energy-used-for/, July 2016

Configuración precisa

Cualquier producto utilizado hoy en día es probable que funcione sin problemas bajo su configuración por defecto. A pesar de ello, ejecutar un software en una plataforma con ajustes de configuración por defecto (independientemente de si se basa en un servidor, un contenedor o utiliza métodos sin servidor) es como llevar unos zapatos sin atar los cordones; puede que queden bien, pero es muy poco probable que se ajusten correctamente. Del mismo modo, las configuraciones de plataforma mal ajustadas son otra forma de desperdiciar energía. Por supuesto, depende de la plataforma y de la flexibilidad que haya para ajustar las configuraciones, pero es conveniente al menos entender las opciones y las implicaciones de utilizar los ajustes por defecto.

Por ejemplo, al considerar las opciones de configuración puede que sea posible descubrir comunicaciones de red ineficientes o subóptimas; ¿quizás nunca se activaron los ajustes de compresión HTTP2 y/o gzip? Del mismo modo, es posible que una máquina virtual Java tenga dificultades para gestionar el Garbage Collection (liberación de recursos de memoria) debido a una asignación de memoria insuficiente. ¿Quizás hay tanta información moviéndose dentro de las cabeceras (la parte inicial de una comunicación que lleva la información técnica de cada mensaje) a través de las redes internas o incluso externas, que los niveles de transmisión son casi el doble de lo que necesitan ser?

Por desgracia, es muy raro que los desarrolladores estudien estos temas en profundidad hasta que surge un problema de rendimiento, y cuando lo hacen, puede que sólo estén en condiciones de hacer “ajustes” para aliviar el problema actual. Tener que resolver rápidamente el problema es entonces una oportunidad perdida para conseguir algo mejor. El desarrollo sostenible de software debe vigilar de cerca las capacidades de la plataforma, no sólo en el momento del lanzamiento, sino también durante toda la vida útil de una aplicación. A menudo se dice que “si no está roto, no lo arregles”, pero por el bien común, a veces tiene sentido desafiar este primer principio de la informática.

Métricas holísticas

Como se ha señalado anteriormente, es fundamental centrar los esfuerzos en las áreas adecuadas y lograr el equilibrio correcto entre el tiempo y los esfuerzos invertidos, por un lado, y las mejoras de rendimiento y usabilidad, por otro. El reto es que a menudo no se dispone de toda la información necesaria y precisa, por lo que las decisiones deben tomarse sobre la base de promedios y proyecciones. Por lo tanto, el primer paso debería ser siempre utilizar los datos disponibles, pero a largo plazo se necesitarán mejores herramientas y métricas para perfeccionar la estrategia.

Independientemente de si los servidores están en las instalaciones o forman parte de un acuerdo en el cloud, ya son un área de interés para los esfuerzos de reducción de energía. Esto es especialmente relevante para los proveedores de servicios cloud, que son propietarios de enormes infraestructuras de servidores y, por tanto, pueden obtener mayores ahorros de cualquier mejora que logren. Esto les permite dedicar importantes esfuerzos (en tiempo y dinero) a investigar nuevas formas de reducir el consumo de energía. Paralelamente, también hacen un seguimiento del impacto ecológico de la energía que consumen para reducir su huella global de CO_2.

Un aspecto que a menudo se pasa por alto al evaluar la eficiencia del sistema es el de la infraestructura “oculta”, es decir, los dispositivos personales. Para muchos, los dispositivos personales sólo son importantes cuando se trata de la satisfacción del cliente. Sin embargo, desde un punto de vista holístico, son muy importantes para los desarrolladores de software porque son prolíficos, están conectados y se suman a la huella energética general.

Por su propia naturaleza, los ordenadores portátiles, las tabletas y los dispositivos inteligentes utilizados serán muy diversos, al igual que los propios usuarios y sus comportamientos individuales, que se combinan para crear un modelo muy complejo. Por supuesto, es fundamental centrarse en la experiencia del usuario y, en concreto, podemos investigar hasta qué punto es “verde” la energía que alimenta los dispositivos del usuario final. Se pueden examinar los costes de la energía en diferentes países para comprender el impacto del derroche de energía en relación con el PIB, los ingresos de los usuarios u otras métricas. También hay que tener en cuenta que el panorama tecnológico de los usuarios del mundo desarrollado es muy diferente del de los países subdesarrollados; incluso en los países desarrollados, el ancho de banda de las redes en las zonas rurales y remotas puede ser muy poco fiable e inestable.

Si la comunidad de desarrolladores puede alinearse en torno a un criterio común y apoyar los objetivos principales de Green Coding, es seguro que se presentarán soluciones innovadoras y formas creativas de abordar los nuevos retos a los que nos enfrentamos a nivel mundial.

Las ventajas de GreenCoding

Una vez que en una organización se ha adoptado una perspectiva de GreenCoding, hemos visto que los principios de GreenCoding pueden alinearse muy bien con los ciclos iterativos y la metodología basada en datos de desarrollo agile. Este enfoque de desarrollo ofrece la oportunidad perfecta para que los principios se asimilen en cada parte del proceso de CICD, transformando así la forma en que desarrolla el software.

Sin embargo, este enfoque del diseño de software está en sus inicios, ya que dentro de la metodología actual de GreenCoding, hay un problema particular que debe ser abordado: puede resultar lento, lo que significaría que los proyectos pueden tener una duración mayor de lo esperado y, por lo tanto, los costes del proyecto podrían aumentar.

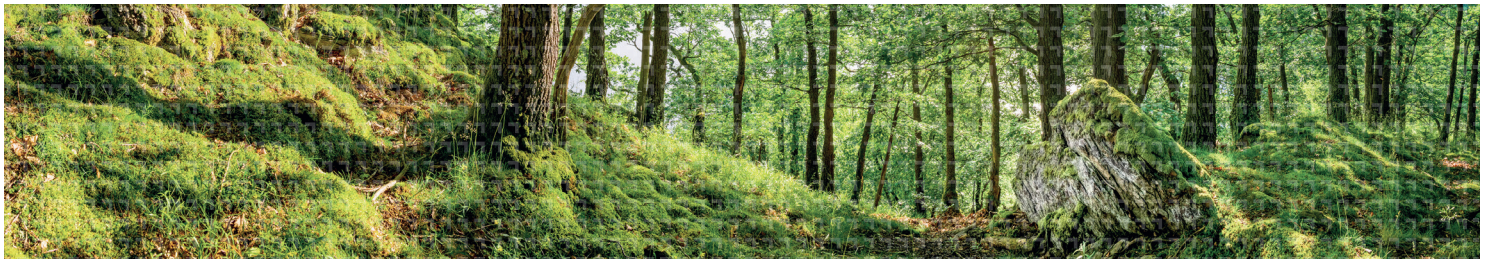
Por ello, este documento está hecho específicamente para cuestionar el pensamiento y las actitudes actuales hacia el desarrollo de software, así como para ofrecer ejemplos de cómo se puede reducir el impacto medioambiental de todo el ciclo de vida del software. Hay que destacar que la emergencia climática sólo puede resolverse mediante una amplia colaboración. Si se adopta un enfoque de colaboración y resolución de problemas y se innovan los procesos actuales de la industria tecnológica y de las TI, corresponde a las organizaciones, a título individual, allanar el camino. Como compañía innovadora en el sector del desarrollo de software, GFT ha esbozado un plan, por el que, al igual que en otras áreas e industrias, GreenCoding tiene el potencial de hacer algo más que reducir las emisiones.

En concreto, el enfoque de GreenCoding para el desarrollo de software puede resumirse de la siguiente manera:

- Mejora el software haciéndolo más eficiente energéticamente también tiene el potencial de hacerlo más fácil de usar y más rápido.
- Mejora la experiencia del usuario y permite ofrecer el software a un público objetivo aún más amplio.
- Los costes operativos del software se vuelven más eficientes, lo que supone un importante ahorro para las empresas.

El concepto de GreenCoding se encuentra todavía en sus inicios e idealmente es algo que se puede poner en práctica con el tiempo; una filosofía que se puede aplicar gradualmente, sin que el tiempo dedicado a los proyectos se vea afectado de forma significativa. Es evidente que el GreenCoding tiene un gran potencial para iniciar un movimiento global entre los desarrolladores, y puede considerarse como una nueva y audaz “frontera verde” para los desarrolladores de software.

Mientras que la eficiencia del cloud computing ha sido un importante catalizador del cambio, en el futuro, GreenCoding es la forma en que se puede hacer todo el software, especialmente en un contexto global en el que todos nos esforzamos por conseguir un planeta sostenible y conectado.



Nuestros autores

Gonzalo Ruiz de Villa Suárez
CTO en GFT

Gonzalo es Director de Tecnología (CTO) de GFT y es responsable de la estrategia de tecnología e innovación de la compañía, incubando iniciativas tecnológicas emergentes a nivel global, liderando proyectos de I+D en el laboratorio de GFT e implementando las últimas innovaciones con los clientes y el ecosistema de socios. Gonzalo es el responsable de la iniciativa GreenCoding en GFT, que crea ideas incrementales, transparentes y de confianza sobre cómo la industria de las TI puede marcar la diferencia para nuestro clima.

Benedict Bax
Asistente Ejecutivo de la CEO en GFT

Benedict es Asistente Ejecutivo de la CEO de GFT y ofrece apoyo en todas las áreas de negocio al equipo directivo global. Su experiencia abarca desde el desarrollo tecnológico hasta la estrategia de negocio, incluyendo un enfoque interdisciplinar con el medio ambiente y la sostenibilidad tecnológica.

Alejandro Reyes Ferreres
Arquitecto de Software en GFT

Alejandro es Arquitecto de Software en GFT y cuenta con un profundo conocimiento tecnológico de las API's y del desarrollo Frontend. Gracias a su experiencia profesional ha desarrollado la iniciativa GreenCoding en GFT ofreciendo ideas originales para su aplicación así como probando los conceptos clave de GreenCoding.

Sobre GFT

GFT impulsa la transformación digital de entidades líderes a nivel mundial en el sector financiero, asegurador e industrial.

Como proveedor de servicios de TI e ingeniería de software, ofrece sus excelentes cualidades en consultoría y en todos los aspectos relacionados con el desarrollo de tecnologías pioneras, como ingeniería cloud, Inteligencia Artificial, modernización de mainframe e IoT para Industria 4.0.

Con su profunda experiencia tecnológica, conocimientos de mercado y sólidas alianzas, GFT implementa soluciones de TI escalables para aumentar la productividad. Esto proporciona a los clientes un acceso más rápido a nuevas aplicaciones de TI y modelos de negocio innovadores, a la vez que reduce los riesgos.

Fundada en 1987 y ubicada en 15 países para garantizar la proximidad a sus clientes, GFT emplea a 6.000 personas a las que ofrece oportunidades profesionales en todas las áreas de ingeniería e innovación de software. Las acciones de GFT Technologies SE cotizan en el Prime Standard de la bolsa de Fráncfort (ticker: GFT-XE). En España, GFT opera desde 2001, donde tiene un equipo de cerca de 2.000 profesionales repartidos entre sus sedes de Alicante, Lleida, Madrid, Sant Cugat (Barcelona), Valencia y Zaragoza.

blog.gft.com/es
twitter.com/gft_es
linkedin.com/company/gft-group
facebook.com/GFT.es
> gft.com/es