Nota: Esta guía está basada en la guía de estilos de Google para proyectos open-source.

Prácticas recomendadas de documentación

> "Diga lo que quiere decir, simple y directamente."

"The Elements of Programming Style", Brian Kernighan

Tabla de Contenidos:

Documentación mínima viable

Un pequeño conjunto de documentos actualizados y precisos son mejores que un conjunto extenso y suelto de "documentación" en varios estados de deterioro.

Escribe documentos cortos y útiles. Elimina todo lo innecesario y mejora cada documento para adaptarse a tus necesidades cambiantes. **Los documentos funcionan mejor cuando están actualizados, pero frecuentemente recortados, como un árbol de bonsai**.

Esta guía anima a los ingenieros a tomar posesión de sus documentos y mantenerlos actualizados con el mismo celo que mantener nuestras pruebas en buen estado. Esfuérzate por esto.

- Identifica lo que realmente necesitas: documentos de lanzamiento, documentos de API, directrices de pruebas.
- Elimina cruft con frecuencia y en pequeños lotes.

Actualiza código y documentos al mismo tiempo

Cambia tu documentación en el mismo CL que el cambio de código. Esto mantiene tus documentos actualizados, y también es un buen lugar para explicar a tu revisor lo que estás haciendo.

Un buen revisor puede al menos insistir en que docstrings, archivos de cabecera, archivos README.md y cualquier otro documento se actualicen junto con el CL.

Elimina la documentación muerta

Los documentos muertos son malos. Desinforman, ralentizan, incitan la desesperación en los ingenieros y la pereza en las pistas del equipo.

Establecen un precedente para dejar atrás los líos en una base de código. Si tu hogar está limpio, la mayoría de los huéspedes estarán limpios sin que se lo pidan.

Al igual que cualquier gran proyecto de limpieza, **es fácil ser abrumado**. Si tus documentos están en mal estado:

- Tómatelo con calma, la salud del documento es una acumulación gradual.
- Primero elimina lo que estés seguro de que está mal, ignora lo que no está claro.
- Involucra a todo tu equipo. Dedica tiempo a escanear rápidamente cada documento y toma una decisión simple: ¿Mantener o eliminar?
- Borra o deja atrás si estás migrando. Los rezagados siempre se pueden recuperar.
- Itera.

Prefiera el bien sobre el perfecto

Tu documentación debe ser tan buena como sea posible pero en un plazo razonable. Los estándares para una revisión de documentación son diferentes de los estándares para revisiones de código. Los revisores pueden y deben pedir mejoras, pero en general, el autor siempre debe ser capaz de invocar la **"Regla de Bien Sobre Perfecto"**. Es preferible permitir que los autores envíen rápidamente cambios que mejoren el documento, en lugar de forzar rondas de revisión hasta que sean "perfectas". Los documentos nunca son perfectos, y tienden a mejorar gradualmente como el equipo aprende lo que realmente necesitan escribir.

La documentación es la historia de tu código

Escribir código excelente no termina cuando su código compila o incluso si su cobertura de prueba alcanza el 100%. Es fácil escribir algo que una computadora entiende, es mucho más difícil escribir algo que un humano y una computadora entiendan. Tu misión como ingeniero consciente de la salud del código es **escribir para los seres humanos en primer lugar, las computadoras en segundo lugar**. La documentación es una parte importante de esta habilidad.

Hay un espectro de documentación de ingeniería que va desde comentarios cortos hasta prosa detallada:

1. **Comentarios en línea**: El propósito principal de los comentarios en línea es proporcionar información que el código en sí no puede contener, por ejemplo, por qué el código está allí.
2. **Comentarios del método y la clase**:
    - **Method API documentation**: Los comentarios de encabezado / Javadoc / docstring que dicen qué hacen los métodos y cómo usarlos. Esta documentación es el contrato de cómo su código debe comportarse. El público objetivo son los futuros programadores que usarán y modificarán su código.

      Es a menudo razonable decir que cualquier comportamiento documentado aquí debe tener una prueba que lo verifica. Esta documentación detalla qué argumentos toma el método, qué devuelve, cualesquiera "gotchas" o restricciones, y qué excepciones puede lanzar o errores que puede devolver. No debes explicar por qué el código se comporta de una manera particular a menos que sea relevante para la comprensión de un desarrollador de cómo utilizar el método. "Por qué" las explicaciones son para comentarios en línea. Piense en términos prácticos al escribir la documentación del método: "Esto es un martillo, lo usas para machacar las uñas".
    - **Class / Module API documentation**: Los comentarios para el encabezado / Javadoc / docstring para una clase o un archivo entero. Esta documentación ofrece un breve resumen de lo que hace la clase / archivo y, a menudo, da algunos ejemplos breves de cómo usar la clase / archivo.

      Los ejemplos son particularmente relevantes cuando hay varias maneras distintas de usar la clase (algunas avanzadas, otras sencillas). Siempre enumere el caso de uso más simple primero.
3. **README.md**: Un buen README.md orienta al nuevo usuario al directorio y apunta a explicaciones más detalladas y guías del usuario:

- ¿Qué se pretende almazenar en este directorio?
- ¿Qué archivos debe buscar primero el desarrollador? ¿Algunos archivos son una API?
- ¿Quién mantiene este directorio y dónde puedo obtener más información?

Consulte las directrices de README.md.