Tema 5
Pruebas de software en métodos ágiles

EI1031 – Verificación y Validación
Grado en Ingeniería Informática

Ramón A. Mollineda Cárdenas

contenidos

- introducción
- e**X**treme **P**rogramming (XP)
  - técnicas XP
  - XP en 12 pasos
  - rol de las pruebas de software en metodología XP
- e**X**treme **T**esting (XT)
  - unit testing
  - acceptance testing
- resumen

introducción

citas sobre innovación y estrategias ágiles …

"It is always wise to look ahead, but difficult to look further than you can see"

W. Churchill (1r ministro UK, 1940-45)

"To improve is to change; to be perfect is to change often"

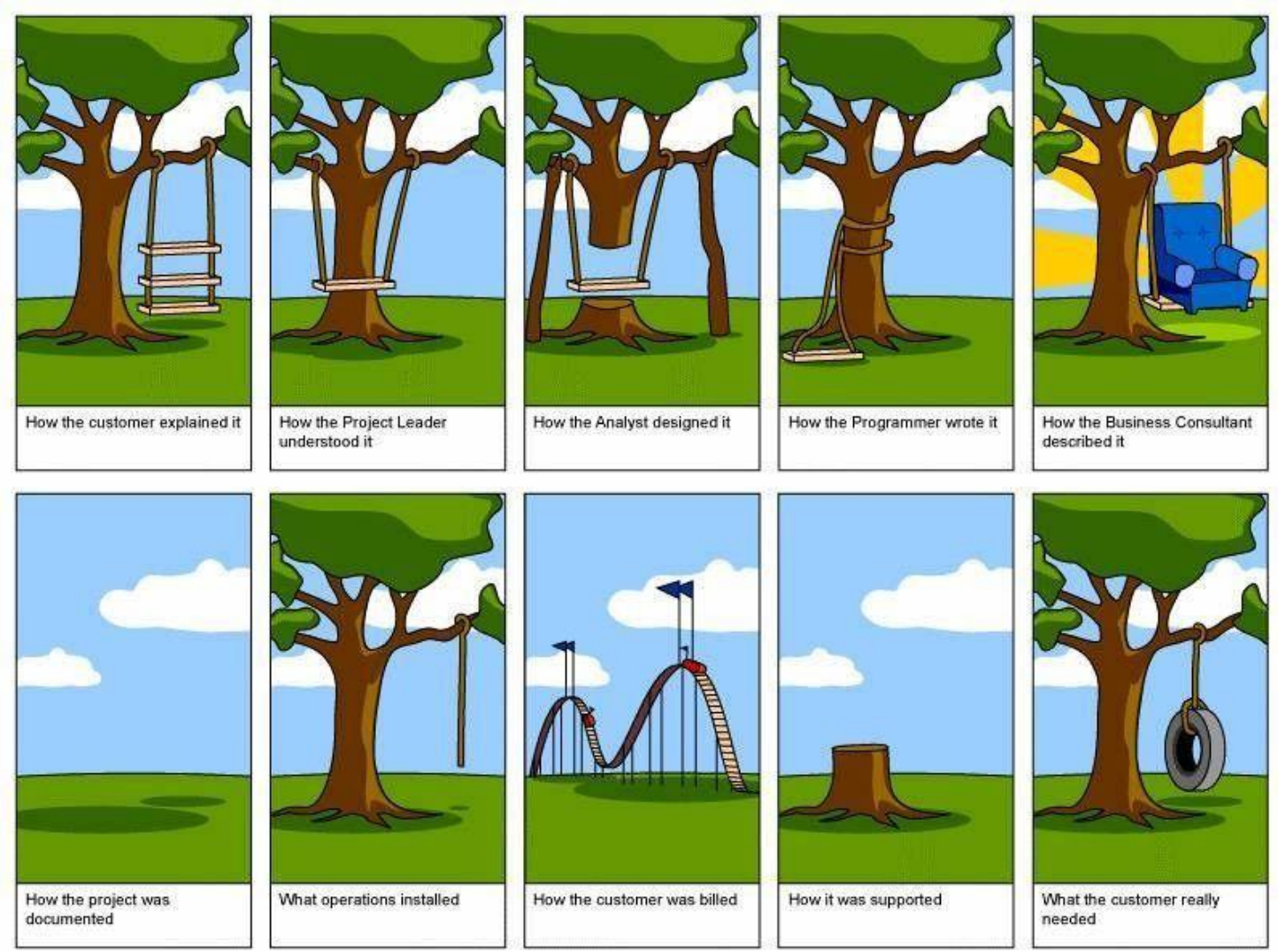
W. Churchill (1r ministro UK, 1940-45)

"Simplicity is the ultimate sophistication"

L. Da Vinci (Polímata, Renacimiento italiano)

introducción

desarrollo clásico



https://www.pinterest.es/pin/497084877592565269/

introducción

desarrollo clásico

¿qué caracteriza el

desarrollo clásico de software?

introducción

desarrollo clásico

el desarrollo clásico (o heavyweight) funciona, pero es muy costoso:

- La planificación y el análisis son procesos muy exhaustivos, que intentan documentar cada característica o tarea del desarrollo completo.
- La comunicación suele realizarse a través de MUCHOS documentos.
- Medir el progreso incluye también documentos terminados.
- Son procesos poco flexibles ante cambios en las especificaciones; cuando se aceptan, el coste (y tiempo) de adaptación es elevado.
- El comportamiento de los individuos está regulado por el proceso (docs); podría no ajustarse a las necesidades del producto.
- Fomenta codificar primero y luego crear y ejecutar pruebas.
- El tiempo de lanzamiento del producto al mercado es grande.

desarrollo ágil: eXtreme Programming

generalidades

XP

metodología/filosofía de desarrollo de software

objetivo de XP

crear rápidamente software con garantías de calidad (productividad)

desarrollo ágil: eXtreme Programming

generalidades

principios de XP

- comunicaciones verbales, honestas, frecuentes y abiertas.
- simplicidad: soluciones simples a necesidades actuales.
- integración continua de nuevas funciones y sus pruebas.
- realimentación frecuente: 1) *sistema* (pruebas de unidad e integración); 2) *cliente* (pruebas de aceptación); 3) *equipo* (reuniones cortas).
- valor para manejar diseños simples, persistir, responsabilizarse, …
- respeto por las decisiones colectivas, por el trabajo de compañer@s, …

desarrollo ágil: eXtreme Programming

técnicas (i)

- planificación y requisitos: especificaciones basadas en user stories; programadores descomponen cada *user story* y hacen estimaciones; clientes escriben *user stories*, priorizan, validan, ofrecen *feedback*, ...
- versiones frecuentes con pequeños incrementos tangibles, funcionales, de valor para el usuario (con base de código estable).
- diseño más simple posible capaz de pasar pruebas; admite cambios.
- metáfora del sistema: forma de entender y comunicar fácilmente el diseño; incluye sistema de nombres de objetos (clases, métodos, ...).
- procesos continuos de pruebas: escribir pruebas antes del código que ha de pasarlas; permite medir progreso; software terminado si pasa *tests*.
- refactoring: cambios en el código + validación con pruebas.

desarrollo ágil: eXtreme Programming

system metaphor

system metaphor

> The system metaphor is a story that everyone (customers, programmers, managers) can tell about how the system works.
>
> Kent Beck (creador de XP y TDD)

ejemplo

> Customer service is an "assembly line".

desarrollo ágil: eXtreme Programming

técnicas (ii)

- pair programming: 2 programadores codifican juntos en un mismo PC => inspección de código en tiempo real => detección y corrección de bugs.
- autoría colectiva del código: ningún programador se dedica exclusivamente a una parte específica del código base.
- integración continua: integración diaria + ejecución automática de pruebas.
- horas extra … ¡prohibidas! 40 h/sem bien aprovechadas deberían ser suficientes; excepción: ocasionalmente, semana previa a una major release.
- despliegue contínuo: validación frecuente de incrementos funcionales por el cliente, resolución rápida de dudas.
- normas de codificación: todo el código debe tener el mismo aspecto.

desarrollo ágil: eXtreme Programming

user story

user story

declaración de un usuario, en lenguaje de negocio, qué hace o necesita hacer como parte de su trabajo.

objetivos

- captura de forma simple y concisa información sobre quién, qué y por qué de un requisito de usuario.
- constituye la base para escribir pruebas de aceptación.

plantilla

"As <a role>, I want <goal/desire> so that <benefit>"

ejemplo

As a user closing the application, I want to be prompted to save anything that has changed since the last save so that I can preserve useful work.

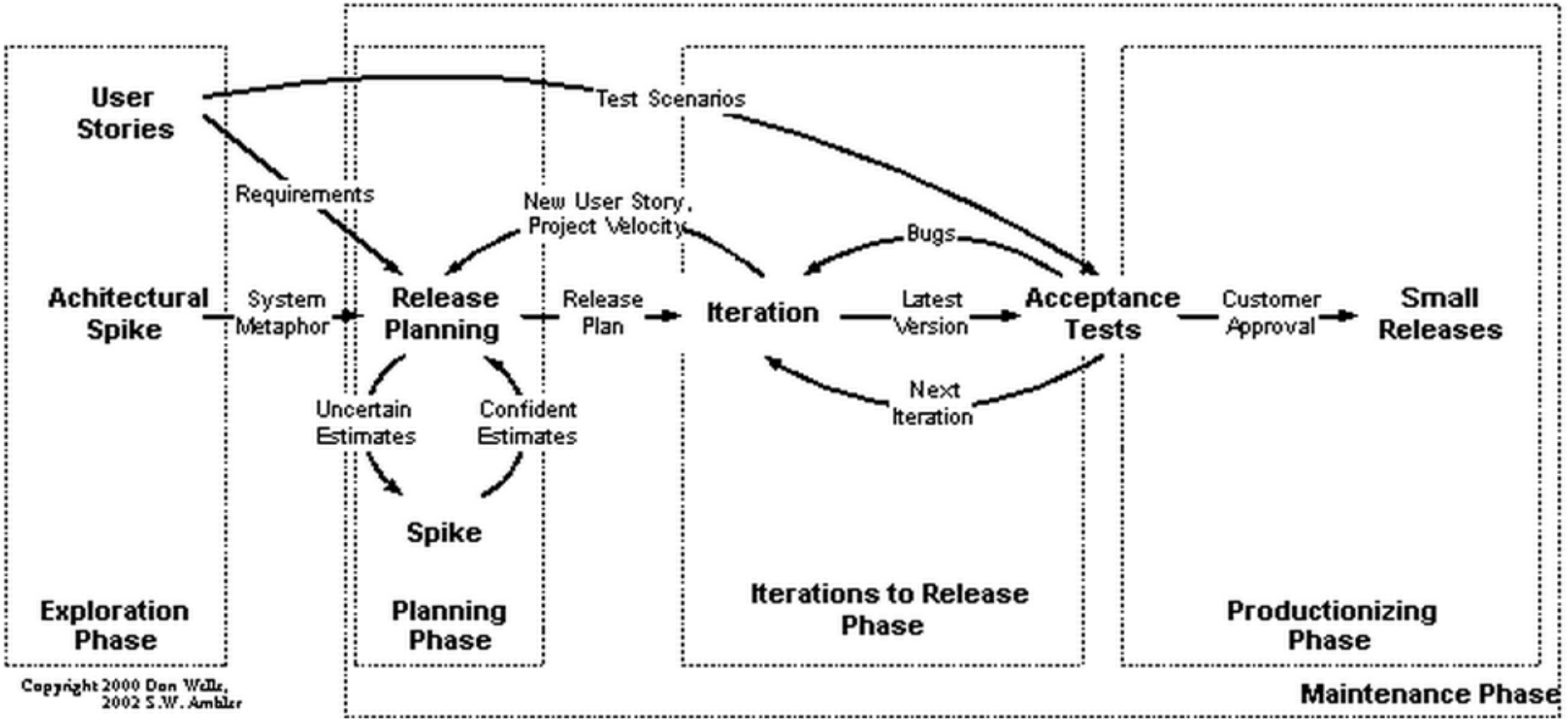
desarrollo ágil: eXtreme Programming

XP en 12 pasos

1. reunión [programadores + clientes] para definir requisitos y *user stories*.
2. reunión [de programadores] para "convertir" requisitos en tareas y estimar.
3. dadas *user stories*, tareas y estimaciones, el cliente crea lista de prioridades.
4. asignación de tareas a parejas de programadores (según habilidades).
5. cada pareja crea pruebas de unidad para su tarea de programación.
6. cada pareja trabaja en su tarea con el objetivo de pasar pruebas.
7. cada pareja mejora su código hasta que pasa todas las pruebas.
8. todas las parejas integran su código base diariamente.
9. el equipo "lanza" una versión incompleta (pre-producción) del sistema.
10. clientes ejecutan pruebas de aceptación => aprobar o informe de mejoras.
11. si pruebas de aceptación OK, el equipo "lanza" una versión de producción.
12. los programadores actualizan estimaciones según experiencia; ir a 3.

desarrollo ágil: eXtreme Programming

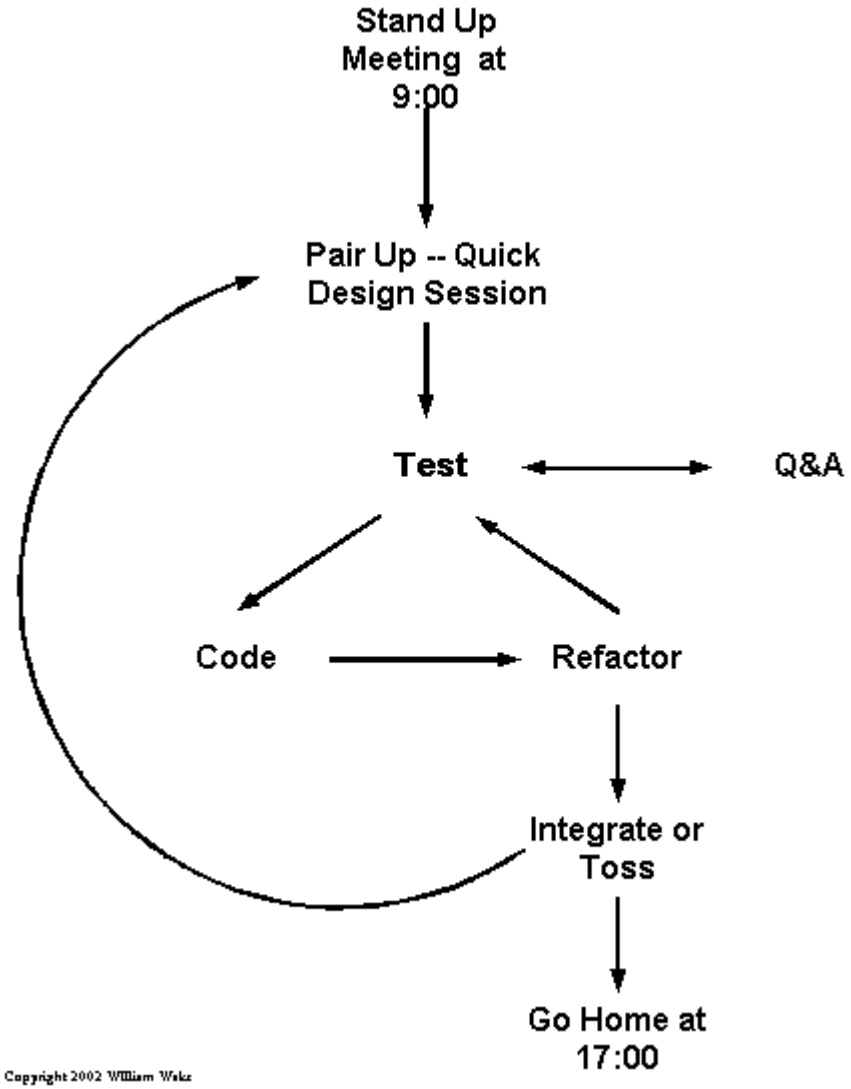
ciclo de vida del método XP



Agile Modeling Throughout the XP Lifecycle.
http://www.agilemodeling.com/essays/agileModelingXPLifecycle.htm

desarrollo ágil: eXtreme Programming

ciclo de vida del método XP



The typical day of an XP Developer.
http://www.agilemodeling.com/essays/agileModelingXPLifecycle.htm

architectural spike

For example, if you were going to write a bank account system, you might spike it by writing a very simple deposit transaction that goes from the user to the database and back again.

Robert C. Martin (American software consultant)
autor de "Clean Code: A Handbook of Agile Software Craftsmanship"

desarrollo ágil: eXtreme Programming

rol de las pruebas de software en metodología XP

los procesos continuos de pruebas son claves en el éxito de **XP** ….

- pruebas de aceptación frecuentes garantizan que las versiones incrementales funcionales satisfacen user stories.
- pruebas unitarias => mayor parte del esfuerzo de pruebas; objetivo: garantizar corrección del código mientras se escribe.
- pruebas constantes dan soporte a refactorización => mejorar, optimizar código base manteniendo funcionalidad.
- pruebas constantes => confianza (beneficio intangible)
  - equipo de desarrollo confía en código base
  - clientes confían en el éxito del proyecto

desarrollo ágil: eXtreme Programming

XP no es la panacea

XP seduce, pero NO es solución universal ….

‡ a favor: implementar proceso de 12 pasos => éxito muy probable.

– en contra: si te saltas algún paso => alto riesgo de fracaso.

– en contra: soluciones simples => cambios futuros podrían costar más que soluciones más generales (y complejas) que prevean cambios.

– en contra: algunos programadores encuentran la programación por pares un proceso invasivo, incómodo, embarazoso, ...

– en contra: el equipo debe tener cultura y experiencia XP.

○ restricciones: XP suele ir bien en proyectos pequeños o medianos, con requisitos cambiantes, si es fácil comunicación directa entre partes.

eXtreme Testing

introducción

XT = ejecución frecuente de pruebas

- XT incluye 2 tipos principales de pruebas: unidad y aceptación.
- NO cambia (w.r.t. a procesos tradicionales de desarrollo) …
  - métodos para elaborar casos/suites de pruebas unitarias
  - el objetivo principal de las pruebas: detectar errores.
- si cambia …
  - la etapa del proceso de desarrollo en el que se escriben.
  - la frecuencia con la que se aplican

eXtreme Testing

unit testing (i)

pruebas de unidad en **XT** deben seguir 2 reglas simples:

- las pruebas de unidad deben ser definidas y escritas antes de escribir el código que ha de pasarlas.
- todas las funciones/métodos deben pasar las pruebas de unidad antes de ser integradas en versiones de producción.

metodología que implementa esta filosofía de pruebas …

- Test-driven development (TDD).
- Test-first programming.
- Test-first development.

eXtreme Testing

unit testing (ii)

¿beneficios de escribir las pruebas antes del código?

eXtreme Testing

unit testing (iii)

pruebas de unidad en **XT**; beneficios:

- comprenderás mejor los requisitos y especificaciones.
- puedes resolver ambigüedades en especificaciones antes de codificar.
- es más probable que tu código satisfaga especificaciones.
- defines formalmente el comportamiento esperado de tu código antes de escribirlo => conoces el objetivo con precisión.
- un diseño inicial simple puede refactorizarse posteriormente sin temor a que deje de cumplir especificaciones.

eXtreme Testing

unit testing (iV)

TDD te ayuda a escribir código a través de pequeños pasos …

motivación:

- dispones de un código functional y probado, aunque incompleto.
- añades nuevo fragmento de código para nuevo caso de uso
- supongamos que, con el nuevo código, introduces errores.
- ¿cómo es más fácil localizar y corregir esos errores?
  - escenario 1: el nuevo fragmento tiene 10 líneas de código
  - escenario 2: el nuevo fragmento tiene 100 líneas de código

eXtreme Testing

unit testing (v)

TDD te ayuda a escribir código a través de pequeños pasos …

principios:

- se asume que existen suites de pruebas automatizadas.
- progresar añadiendo nuevos fragmentos de pocas líneas de código.
- compilar y ejecutar pruebas después de añadir cada fragmento.

eXtreme Testing

acceptance testing (i)

pruebas de aceptación en XT:

qué: satisfacer …

… requisitos funcionales de usuario,

… requisitos operacionales (rendimiento, seguridad, etc.),

… requisitos de legalidad, usabilidad, etc.

quién: creación conjunta por desarrolladores, testers y usuarios.

ejecución manual por usuarios (beta testing), o automática por equipo de desarrollo.

cuándo: creación en fases de planificación y diseño de iteración.

ejecución después de cada versión de preproducción.

eXtreme Testing

acceptance testing (ii)

comentarios:

- las *user stories* son la base para crear pruebas de aceptación.
- una *user story* suele dar lugar a múltiples pruebas de aceptación.
- si un usuario detecta varios bugs, debe crear lista con prioridades; después de corrección, el usuario repite pruebas de aceptación.
- asumamos que las pruebas de unidad son suficientes y correctas, y que el software bajo análisis las pasa **TODAS**; sin embargo, podría fallar en pruebas de aceptación … *¿por qué?*

eXtreme Testing

pruebas como documentación …

pruebas de unidad como documentación del software:

- muchos programadores prefieren documentarse sobre el diseño y el código estudiando ejemplos de uso.
- las pruebas de unidad construyen estructuras de datos e invocan métodos para realizar pruebas, lo cual constituye ejemplos de cómo usar el código.

pruebas de aceptación como definición de requisitos:

- las pruebas de aceptación definen de forma precisa, a través de ejemplos, qué espera un usuario del software (requisitos, *user stories*).
- por lo tanto, estas especifican requisitos de manera formal.

resumen

resumen:

- la competitividad de nuestros tiempos “obliga” a desarrollar rápido e introducir productos en el mercado en poco tiempo
- los modelos tradicionales no dan respuesta a esta necesidad
- el modelo eXtreme Programming (XP) fue concebido para dar soporte al desarrollo rápido de aplicaciones informáticas de calidad
- procesos continuos de pruebas (eXtreme Testing, XT) son claves en XP
- XT se centra en las pruebas de unidad y de aceptación
- las pruebas de unidad se crean antes de codificar
- las pruebas de unidad se ejecutan cada vez que el código cambia

bibliografía

Glenford J. Myers. The Art of Software Testing. 2ª Edición. Wiley, 2004.