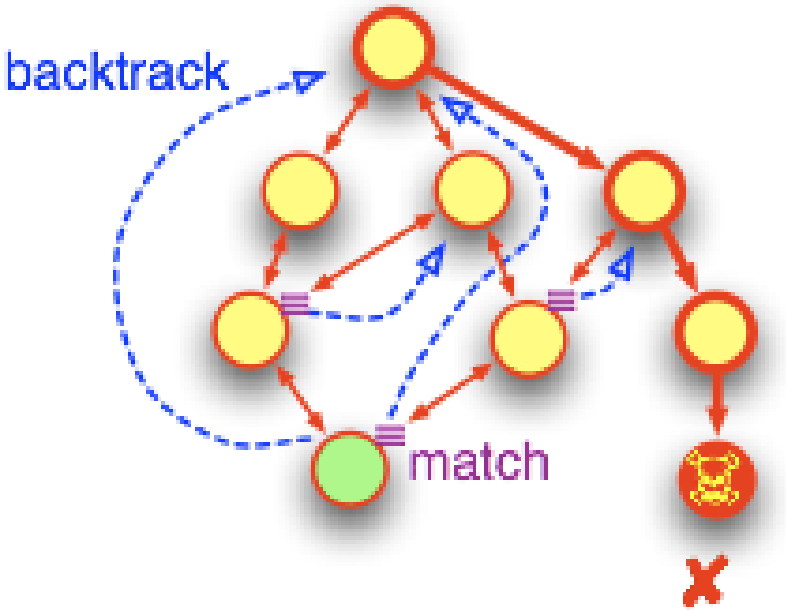
Ingeniería de Software I
Técnicas de Análisis

Jocelyn Simmonds

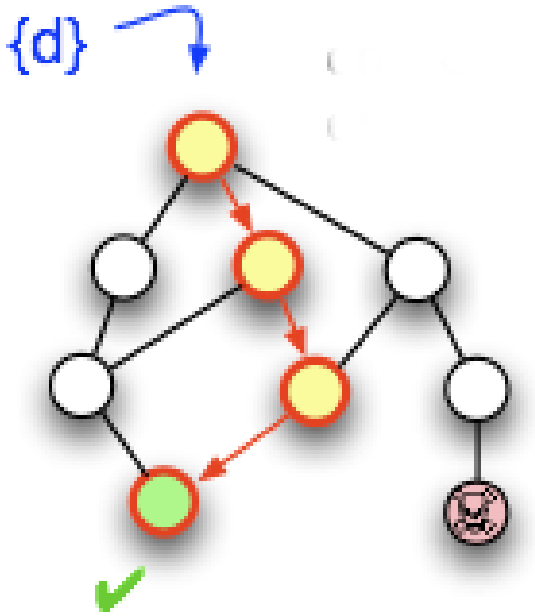
Departamento de Ciencias de la Computación

Análisis vs Testing

Análisis: construir un modelo del sistema y analizar si cumple ciertas propiedades



Testing: ejecutar algunas trazas para ver si se produce un error



Son complementarias, y tienen ventajas y desventajas distintas. En la clase de hoy veremos técnicas **manuales** de análisis de código.

Peligros del acostumbramiento . . .

Leemos el código una y otra vez mientras trabajamos . . .

- . . . y nos vamos "acostumbrando" al código
- cuando conoces el código demasiado bien, ya no se ven los errores
- alguien con ojos "frescos" debe revisar el código

Técnicas manuales de análisis

- Revisiones manuales de código
  - Walkthroughs: revisión informal, paso a paso del código
  - Revisiones técnicas (technical reviews): autor del artefacto guía la reunión
  - Inspecciones: proceso formal, guiado por un moderador
- Predicción de defectos (defect prediction)

Cuando usar revisiones manuales de código ...

- Son aplicables en cualquier etapa del proceso de desarrollo
- Han mostrado aumentar la confiabilidad del software, pero ...
  - son técnicas intensivas en HH
  - usualmente es lo primero que deja de hacerse cuando hay poco tiempo
  - y a menudo se hacen de manera informal, sin datos históricos, en forma no repetible

Walkthroughs

Revisión usualmente informal:

- el autor de un artefacto de software guía al resto del equipo en una revisión paso a paso del artefacto
  - artefacto de software: diseño detallado, código, casos de uso, proceso de negocio, especificación de casos de prueba, etc.
  - autor: diseñador, programador, etc.
- objetivos usuales de este tipo de revisión
  - obtener feedback acerca de la calidad o contenido del artefacto
  - familiarizar al equipo con el artefacto a revisar

Revisiones técnicas

Más formal que un walkthrough:

- objetivo: analizar la calidad técnica de un artefacto de software para mejorarlo
  - artefacto de software: igual que en la diapo anterior
- posibles mejoras: corrección de defectos, recomendación de un enfoque alternativo
  - ¿por qué haces X en linea Y? ¿has probado la biblioteca Z para hacer W?
- requiere más preparación que un walkthrough, participantes no pueden llegar "en blanco"

Roles en una revisión técnica

- review leader: usualmente el autor del artefacto siendo revisado. Es el encargado de tareas administrativas asociadas a la revisión, y de guiar la revisión para ver que se cumplan los objetivos de esta
- decision maker: determina si los objetivos de la revisión fueron cumplidos
- recorder: alguien que documente los bugs, acciones a realizar, decisiones y recomendaciones hechas en la revisión
- equipo técnico: participantes activos de la revisión y evaluación del artefacto

También se puede invitar a cliente y/o usuarios, depende del artefacto a revisar.

Inspecciones (M. Fagan, 1972)

Más formal que una revisión técnica:

- objetivo: identificar defectos en un artefacto de software
- artefactos típicos: código, especificación de requerimientos, planes de pruebas
- proceso general:
  - los "inspectores" deben estudiar el artefacto antes de la reunión, identificando posibles errores
  - errores dependen del tipo de artefacto: error en código, inconsistencia en los requisitos, caso borde no testeado, etc.

Fases típicas de una inspección (1/2)

- Planning:
  - el autor del artefacto deben preparar documentos para los inspectores, explicando el contenido de estos
  - el moderador revisa estos documentos, consigue participantes, les entrega el material y agenda reunión
- Overview meeting:
  - el autor del artefacto describe este más cualquier otra información relevante para los inspectores
- Preparation:
  - cada inspector debe examinar al artefacto en detalle, identificando posibles defectos

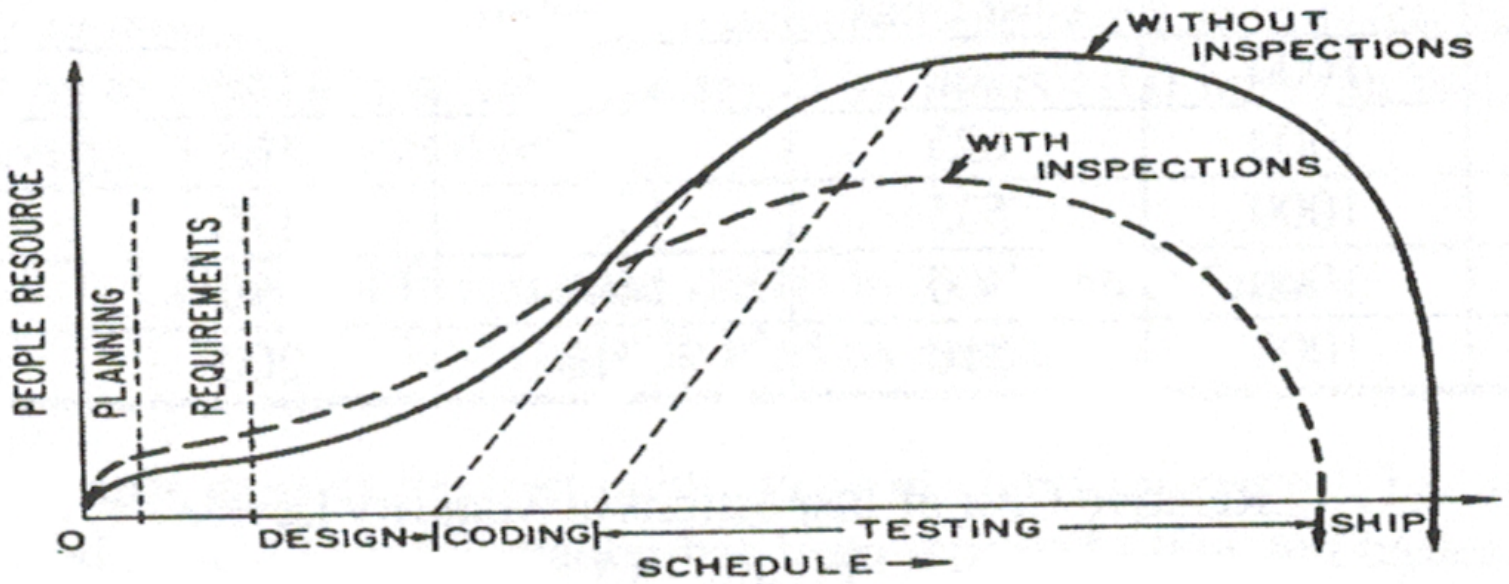
Fases típicas de una inspección (2/2)

- Inspection meeting:
  - se examina el artefacto paso a paso, donde los inspectores van indicando los defectos que detectaron
  - también se pueden detectar nuevos defectos
  - NO deben discutir soluciones alternativas
  - ¡critiquen al artefacto, y no al autor de este!
- Rework:
  - el autor del artefacto debe corregirlo de acuerdo a las acciones determinadas en la reunión de inspección
- Follow-up:
  - el equipo revisa los cambios hechos por el autor del artefacto

Roles en una inspección

- autor: la persona que creo el artefacto que va a ser inspeccionado
- moderador: planifica y guía el proceso de inspección
- lector: la persona que va "leyendo" el artefacto durante la reunión de inspección
- inspectores: personas que examinan el artefacto y su documentación para encontrar posibles defectos
- recorder: alguien que documente los defectos encontrados en la inspección

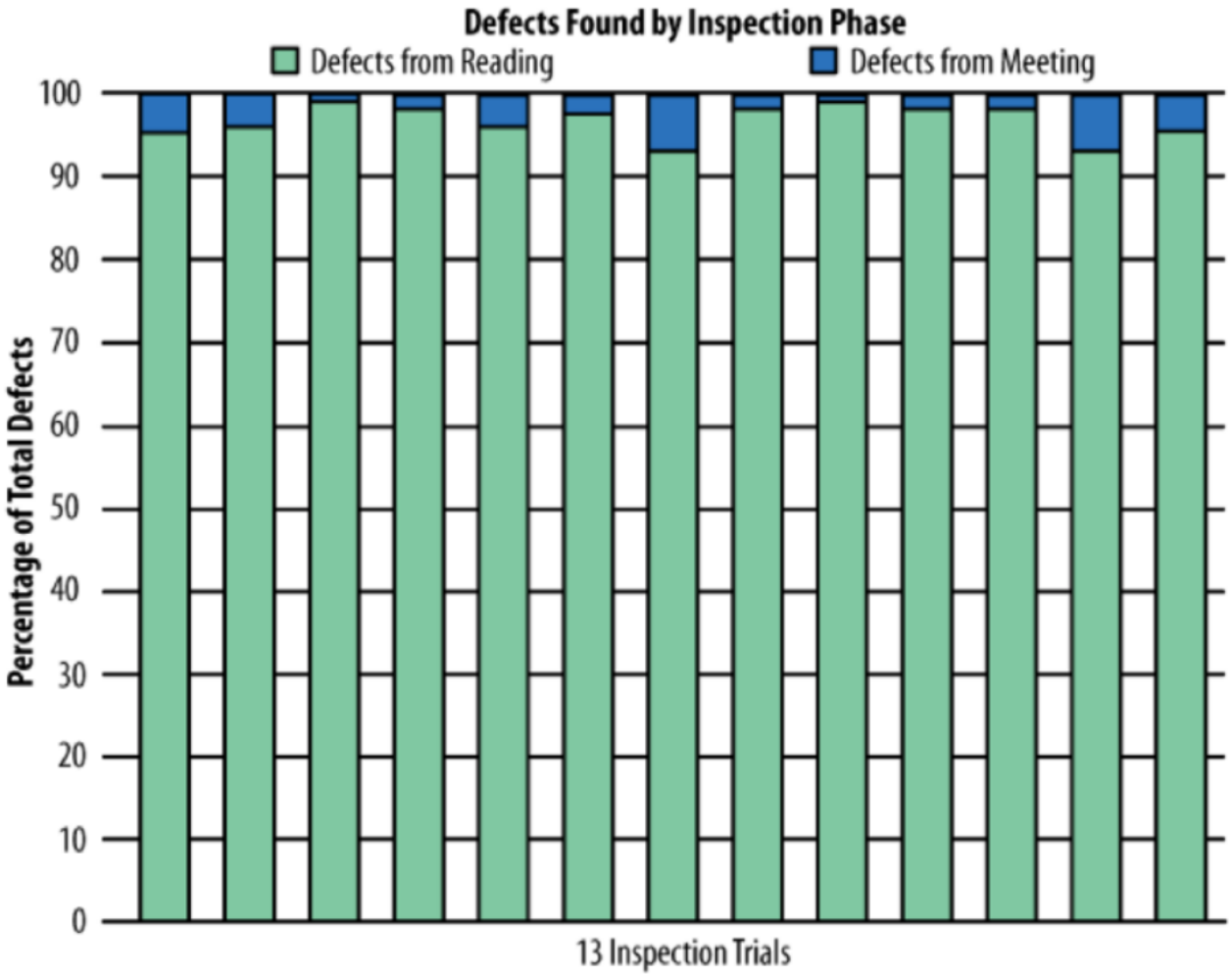
Efectividad de inspecciones



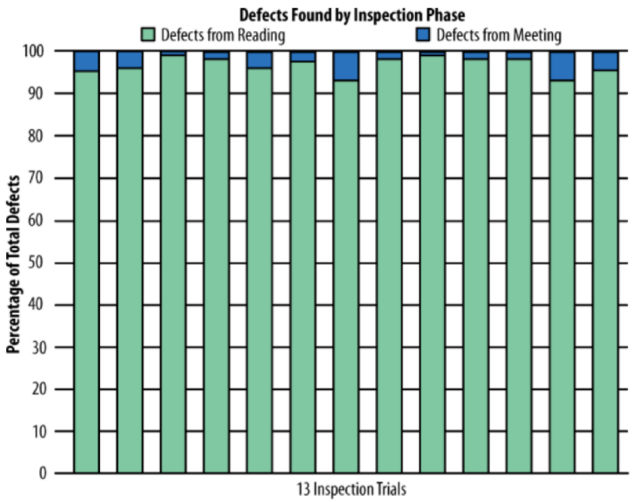
Fagan estudio en forma empírica el uso de inspecciones en IBM (1986)

- las personas duplicaron la cantidad de lineas de código que produjeron
- se encontraron 60 - 90% de los defectos
- los defectos se encontraron cerca de donde fueron introducidos, lo que reduce el costo de corrección

¿Cuándo se encontraron los defectos?



¿Cuándo se encontraron los defectos?



Casi el 95% de los defectos se encontraron afuera del inspection meeting

- la mejor forma de encontrar errores lógicos sutiles es leyendo el código
- las reuniones (HH) sirven para filtrar falsos positivos . . .
- . . . y se podría hacer por email

Más datos

- Mills (1990) encontró que un 90% de los defectos se podían encontrar con inspecciones
- McConnell (1993) comparo la efectividad de unit testing e inspecciones: inspecciones logro detectar 60% de los defectos, unit testing solo 25%
- ¿Por qué las inspecciones son tan efectivas?

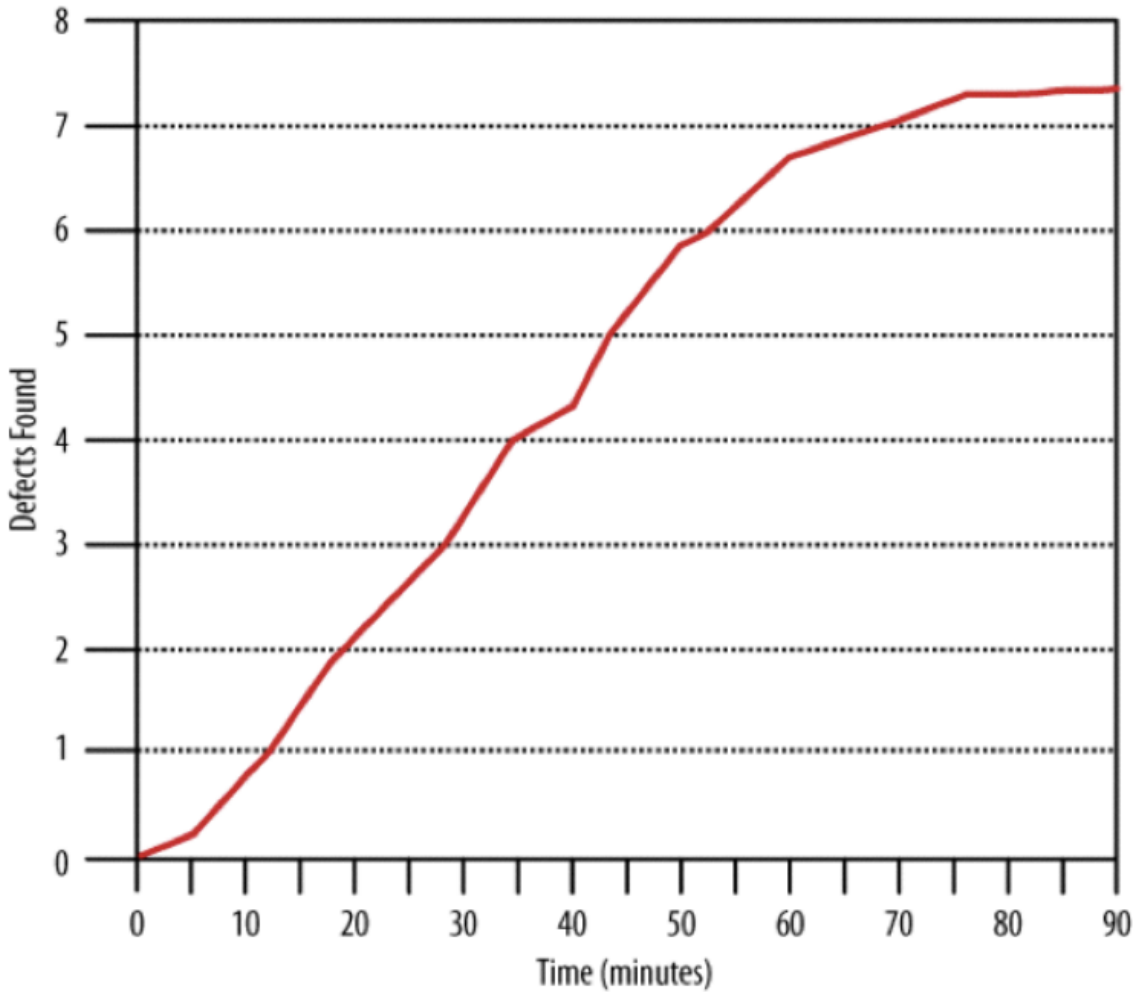
Más datos

- Mills (1990) encontró que un 90% de los defectos se podían encontrar con inspecciones
- McConnell (1993) comparo la efectividad de unit testing e inspecciones: inspecciones logro detectar 60% de los defectos, unit testing solo 25%
- ¿Por qué las inspecciones son tan efectivas?
  - al saber que otros van a examinar sus artefactos, los desarrolladores son más cuidadosos, lo que lleva a un mejor producto
  - tener más personas escudriñando un artefacto aumenta la probabilidad de que se encuentren defectos
- Las revisiones técnicas y walthroughs también parecen ser efectivas, pero es mas difícil hacer estudios empíricos al respecto

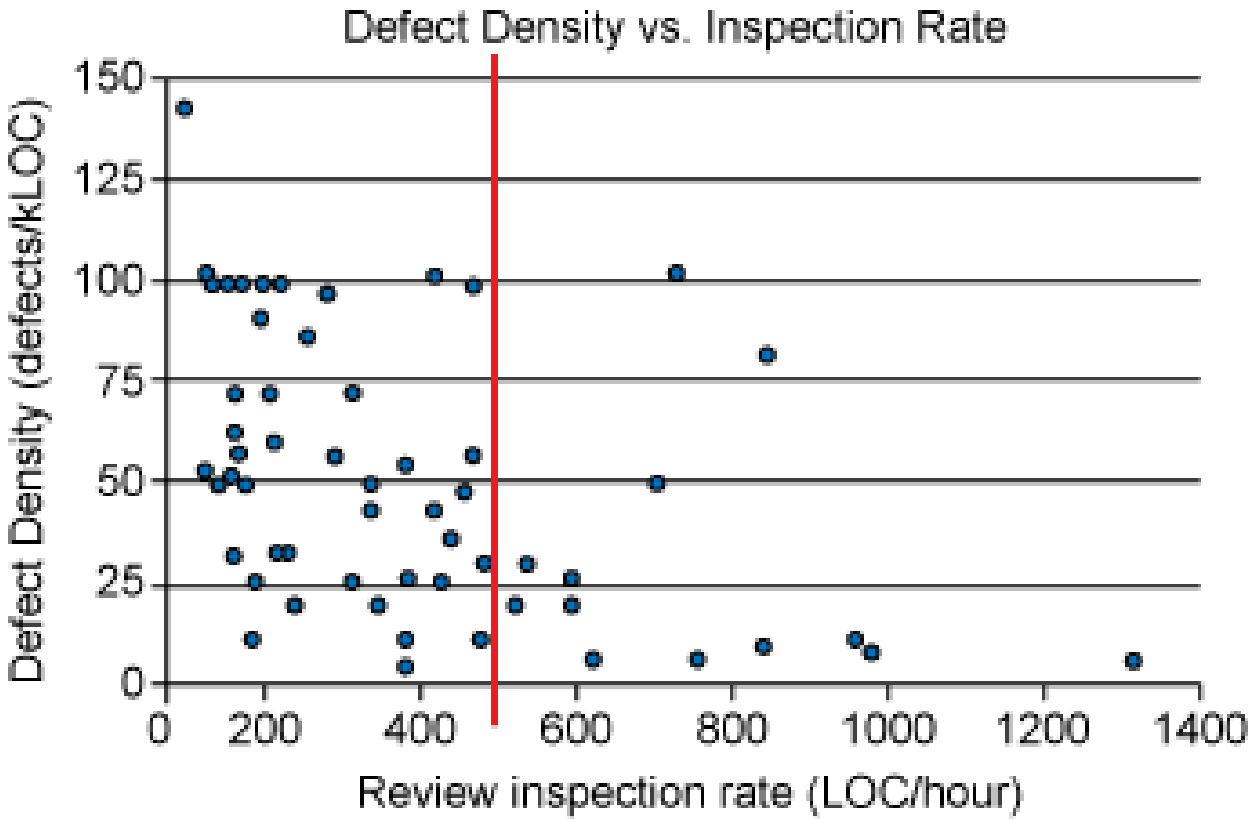
¿Duración "óptima" para una revisión manual?

Estudios muestran:

- que se puede encontrar un defecto cada 10 minutos ...
- ...con un tope de 60 minutos



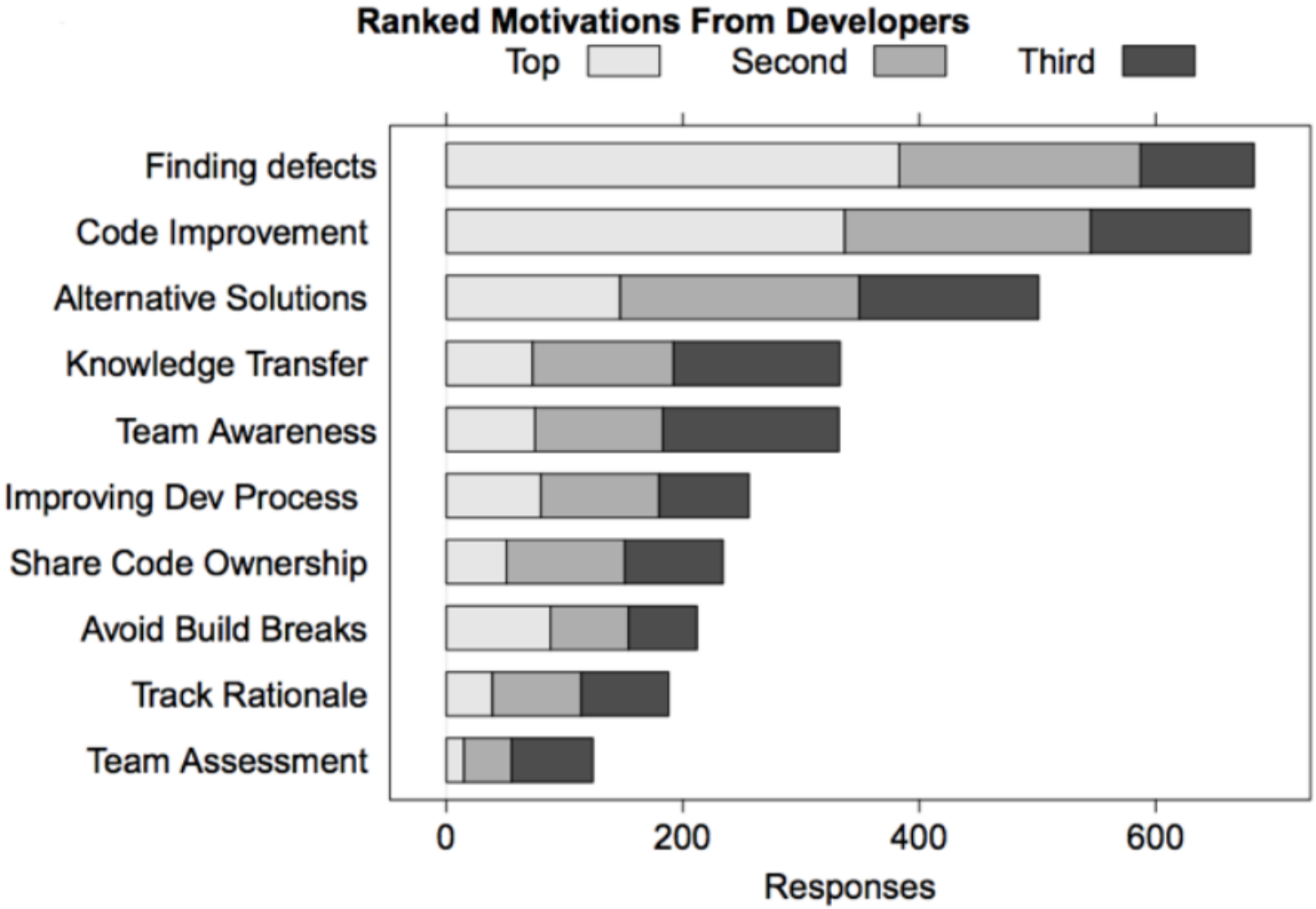
¿"Velocidad" de lectura?



La "densidad" de defectos empieza a caer a partir de las 400 LOC por hora . . . limitar las revisiones a una hora y $\approx$ 300 LOC.

Motivación para hacer revisiones manuales

Resultados de entrevistas con desarrolladores en Microsoft (2012):



¿Y qué pasa si . . . ?

- "Nosotros trabajamos en forma ágil . . . "

¿Y qué pasa si . . . ?

- "Nosotros trabajamos en forma ágil . . ."
  - no es habitual hacer revisiones formales del código
  - pero pair programming es una forma de hacer revisión del código
- "Trabajamos en forma distribuida . . .", "Somos una empresa demasiado grande . . ."

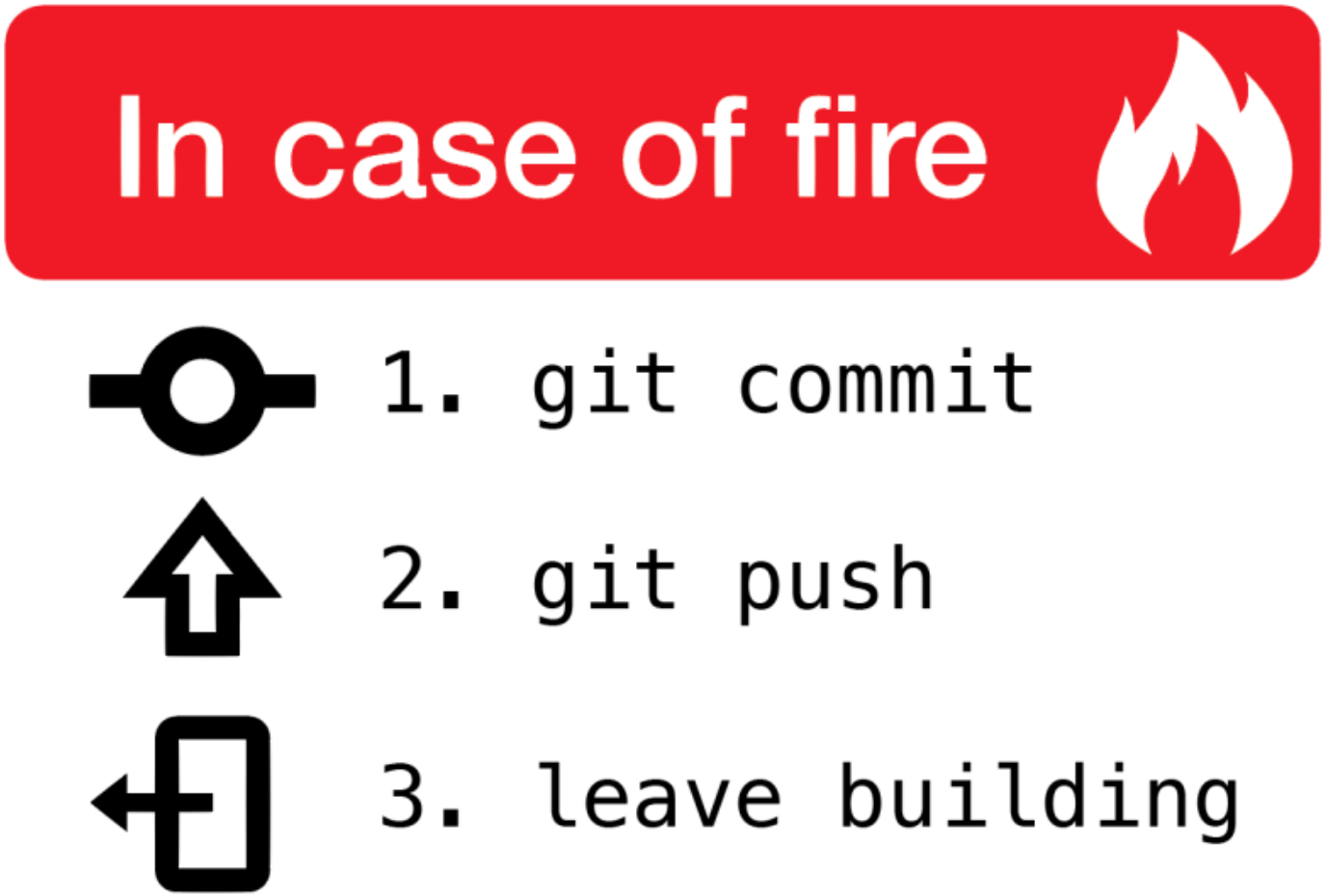
¿Y qué pasa si . . . ?

- "Nosotros trabajamos en forma ágil . . ."
  - no es habitual hacer revisiones formales del código
  - pero pair programming es una forma de hacer revisión del código
- "Trabajamos en forma distribuida . . .", "Somos una empresa demasiado grande . . ."
  - establecer una política de "check at check in"
  - cada cambio debe ser revisado por una persona antes de que sea aceptado
  - ver `https://git-scm.com/book/it/v2/Distributed-Git-Distributed-Workflows`

Check at check in

Algunas cosas a revisar antes de hacer check in:

- adherencia al coding style
  - `https://github.com/torvalds/linux/blob/master/Documentation/process/coding-style.rst`
- uso de variables no inicializadas
- que existan bucles infinitos
- que hayan índices de arreglos fuera de los límites
- que haya división por cero
- que la memoria sea correctamente asignada/liberada
- que las excepciones se hayan manejado correctamente
- ...


In case of fire
1. git commit
2. git push
3. leave building


BUILD IS BROKEN
PUSH ANYWAYS
memegenerator.net

Recomendaciones acerca de revisiones

- Revisen a lo más 300 LOC a la vez, en bloques de 1 hora
- Es probable que las inspecciones formales sean innecesarias
- Revisar nuestro propio codigo es mejor que nada, pero ojala combinar con algun tipo de revision externa
- Usen revisiones manuales para "distribuir" el conocimiento acerca del código fuente

Predicción de defectos

Los sistemas son cada vez más grandes . . .

- . . . no podemos revisar todos los archivos

Predicción de defectos

Los sistemas son cada vez más grandes . . .

- . . . no podemos revisar todos los archivos
- pero si tenemos datos historicos acerca del sistema (versionamiento, issue tracking, etc.) . . .
- podemos construir modelos de predicción de defectos para determinar cuales archivos debemos revisar con más cuidado

Predicción de defectos

Los sistemas son cada vez más grandes . . .

- . . . no podemos revisar todos los archivos
- pero si tenemos datos historicos acerca del sistema (versionamiento, issue tracking, etc.) . . .
- podemos construir modelos de predicción de defectos para determinar cuales archivos debemos revisar con más cuidado

Que nos interesa:

- cuales archivos tienen defectos
- como rankear los archivos por numero probable de defectos

Los defectos no están distribuidos en forma uniforme

Datos de AT&T:

System	Final release files	Final release KLOC	% faults in top 20% files
Inventory	1950	538	83%
Provisioning	2308	438	83%
Voice Response	1888	329	75%
Maintenance Support A	668	442	81%
Maintenance Support B	1413	384	93%
Maintenance Support C	584	329	76%

Métricas

Las siguientes son métricas que podemos usar para construir modelos de predicción de defectos:

- cobertura del código
- code churn (cuanto código cambio en un artefacto)
- complejidad del código
- dependencias en el código
- métricas acerca de las personas y la organización

Podemos combinar métricas, y realizar predicciones a nivel de binarios, paquetes, componentes, archivos, clases, funciones, etc.

Predicción de defectos en la practica

Google trata de detectar "hot spots", archivos que hay que revisar con mas cuidado:

> *. . . files are flagged if they have attracted a large number of bug-fixing commits, no more and no less. . . . creating a program that hooked into our source control system, and pulls out all the changes which had a bug attached to them. . . . For each file, the number of bug-fixing changes it's been in is calculated, and we output the files which were ranked in the top 10%.*

Lean el siguiente post: `http://google-engtools.blogspot.cl/2011/12/bug-prediction-at-google.html`