

# Gestión de Calidad en el Software

Ingeniería en Sistemas de Información

(IS)GCS-M1: Calidad de Productos y Procesos

Profesor: Gerardo I. Simari

Depto. de Ciencias e Ingeniería de la Computación

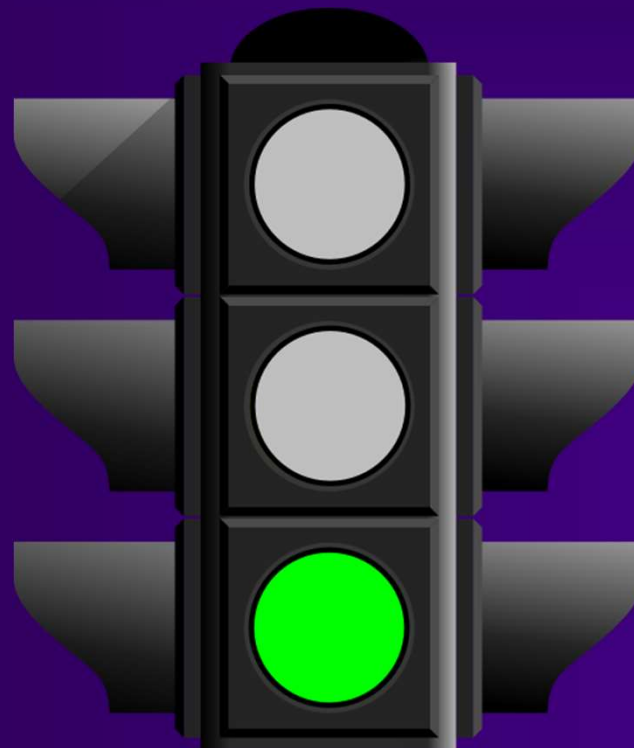
Universidad Nacional del Sur – Bahía Blanca, Argentina

1er. Cuatrimestre de 2020



*Ejercicio con autoevaluación:*

*Continuar para realizar una autocorrección*



# Checklist para Definición de Proceso

- 1) Diferenciar claramente los conceptos de *Entrada*, *Salida* y *Activos*
  - ¿Los activos enunciados son elementos de ayuda a la ejecución del proceso? (Métodos, *templates*, guías, herramientas.)
  - ¿Las entradas indicadas representan elementos utilizados durante el proceso?
  - ¿Hay consistencia “por nombre” con los elementos de entrada y los utilizados por el proceso en su descripción de pasos?
  - ¿Hay elementos que se utilicen en el proceso como información y no estén enunciados entre los elementos de entrada?
  - ¿Todas las salidas enunciadas son generadas por el proceso?

# Checklist para Definición de Proceso

- 1) Diferenciar claramente los conceptos de *Entrada*, *Salida* y *Activos*
  - ¿Hay *consistencia* por nombre con los elementos de salida y los generados por el proceso?
  - ¿Todos los elementos *generados* por el proceso están identificados en las salidas?
  - ¿Todos los activos enunciados están siendo *utilizados* por el proceso?
  - ¿Hay *consistencia* por nombre entre los activos enunciados y los utilizados en la descripción de pasos del proceso?
  - ¿Todos los *activos* utilizados por el proceso están indicados como activos?

# Checklist para Definición de Proceso

## 2) Diferenciar claramente los conceptos de Entrada, Salida, *Precondiciones* y *Postcondiciones*

- ¿Todas las entradas enunciadas representan elementos/información *necesaria* para la ejecución del proceso?
- ¿Todas las salidas enunciadas representan elementos/información *generados* durante el proceso?
- ¿Toda precondición enunciada representa un estado en el contexto que debe verificarse como *válido* para que el proceso se pueda ejecutar?

# Checklist para Definición de Proceso

## 2) Diferenciar claramente los conceptos de Entrada, Salida, *Precondiciones* y *Postcondiciones*

- ¿Están enunciadas todas las *precondiciones necesarias* para la ejecución del proceso?
- ¿Toda postcondición enunciada representa un estado que *cambió* en el contexto luego de la ejecución del proceso?
- ¿Están enunciadas todas las *postcondiciones necesarias* para la ejecución del proceso?

# Checklist para Definición de Proceso

## 3) Respetar convenciones de diagramas de flujo

- ¿Se respeta una *notación* estándar?
- ¿El diagrama de flujo es *válido*?
  - ¿Todos los condicionales tienen las salidas enunciadas para todas las alternativas?
  - ¿Cada flujo de salida del condicional está etiquetado con la alternativa que representa?
  - ¿Todas las actividades están identificadas con un verbo?
  - ¿El diagrama identifica claramente dónde inicia y dónde termina?
  - ¿Todos los caminos posibles conducen al fin del proceso?

# *Checklist* para Definición de Proceso

## 3) Respetar convenciones de diagramas de flujo (cont.)

- En caso de no utilizar una notación estándar, ¿se definieron las referencias para cada símbolo empleado?
- Si se trata de un diagrama funcional:
  - ¿Están claramente identificadas las actividades que responden a cada rol?
  - ¿Todo rol participante en el proceso tiene su banda asignada en el diagrama?



# Checklist para Definición de Proceso

## 4) Consistencia del proceso

- ¿Se define claramente el *objetivo perseguido* con la definición del proceso?
- ¿Está claramente identificado como primer paso detallado del proceso el evento que *dispara* la ejecución del mismo?
- ¿Toda actividad enunciada en la descripción de pasos tiene *trazabilidad* directa con una actividad del diagrama que lo representa?
- ¿Toda actividad representada en el diagrama tiene *trazabilidad* directa con un paso detallado descrito en el proceso?
- ¿Todas las componentes estudiados para un proceso están definidas?

# Checklist para Definición de Proceso

## 4) Consistencia del proceso (cont.)

- ¿Todas las entradas y salidas enunciadas están representadas en el diagrama *asociadas* a las actividades que las usan y/o generan?
- ¿Están enunciados todos los *roles* participantes del proceso?
- ¿Toda actividad descrita en el detalle de pasos identifica el *rol* que la debe ejecutar?
- ¿Está identificado el *owner* del proceso?
- Si usted le diera el proceso a una persona que recién ingresa a la empresa, ¿considera que con la descripción realizada puede llevar a cabo las tareas sin ningún tipo de ambigüedad?

Si su respuesta es NO, *refine* la descripción.

# Agenda: Parte II

- Caso de estudio: QA en Microsoft (1980s a la actualidad)
- QA en procesos de desarrollo de SW: Introducción
  - Buenas prácticas
  - Manejo de *issues* (problemas/asuntos)
    - Rastreo
    - Aplicación/Conformidad
    - Reporte

# Caso de estudio:

## *Quality Assurance* en Microsoft

# Caso de estudio: QA en Microsoft

Pantalla ctrl-alt-del en Windows 3.1 (circa 1992-1993)

**Contoso Deluxe Music Composer**

**This Windows application has stopped responding to the system.**

- \* Press ESC to cancel and return to Windows.**
- \* Press ENTER to close this application that is not responding. You will lose any unsaved information in this application.**
- \* Press CTRL+ALT+DEL again to restart your computer. You will lose any unsaved information in all applications.**

# Caso de estudio: QA en Microsoft

BSOD (*Blue Screen of Death*) en Windows 9x (1995–2000)

Windows

An error has occurred. To continue:

Press Enter to return to Windows, or

Press CTRL+ALT+DEL to restart your computer. If you do this,  
you will lose any unsaved information in all open applications.

Error: 0E : 016F : BFF9B3D4

Press any key to continue \_

# Caso de estudio: QA en Microsoft

BSOD (*Blue Screen of Death*) en Windows 2000 (1999–2005)

```
*** STOP: 0x0000007B (0xF201B84C,0xC0000034,0x00000000,0x00000000)  
INACCESSIBLE_BOOT_DEVICE
```

If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:

Check for viruses on your computer. Remove any newly installed hard drives or hard drive controllers. Check your hard drive to make sure it is properly configured and terminated. Run CHKDSK /F to check for hard drive corruption, and then restart your computer.

Refer to your Getting Started manual for more information on troubleshooting Stop errors.



# Caso de estudio: QA en Microsoft

BSOD (*Blue Screen of Death*) en Windows Vista (2006–2009)

A problem has been detected and Windows has been shut down to prevent damage to your computer.

DRIVER\_IRQL\_NOT\_LESS\_THAN\_OR\_EQUAL\_TO

If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup options, and then select Safe Mode.

Technical information:

\*\*\* STOP: 0x000000D1 (0x00000000, 0x00000000)



# Caso de estudio: QA en Microsoft

BSOD (*Blue Screen of Death*) en Windows 7 (2009 a hoy)

A problem has been detected and Windows has been shut down to prevent damage to your computer.

## MEMORY\_MANAGEMENT

If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

## Technical Information:

\*\*\* STOP: 0x0000001A

Beginning dump of physical memory  
Physical memory dump complete.

Contact your system administrator or technical support group for further assistance.

# Caso de estudio: QA en Microsoft

BSOD (*Blue Screen of Death*) en Windows 10 (2015 a hoy)



Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

20% complete



For more information about this issue and possible fixes, visit <https://www.windows.com/stopcode>

If you call a support person, give them this info:

Stop code: CRITICAL\_PROCESS\_DIED

# La cultura de Microsoft

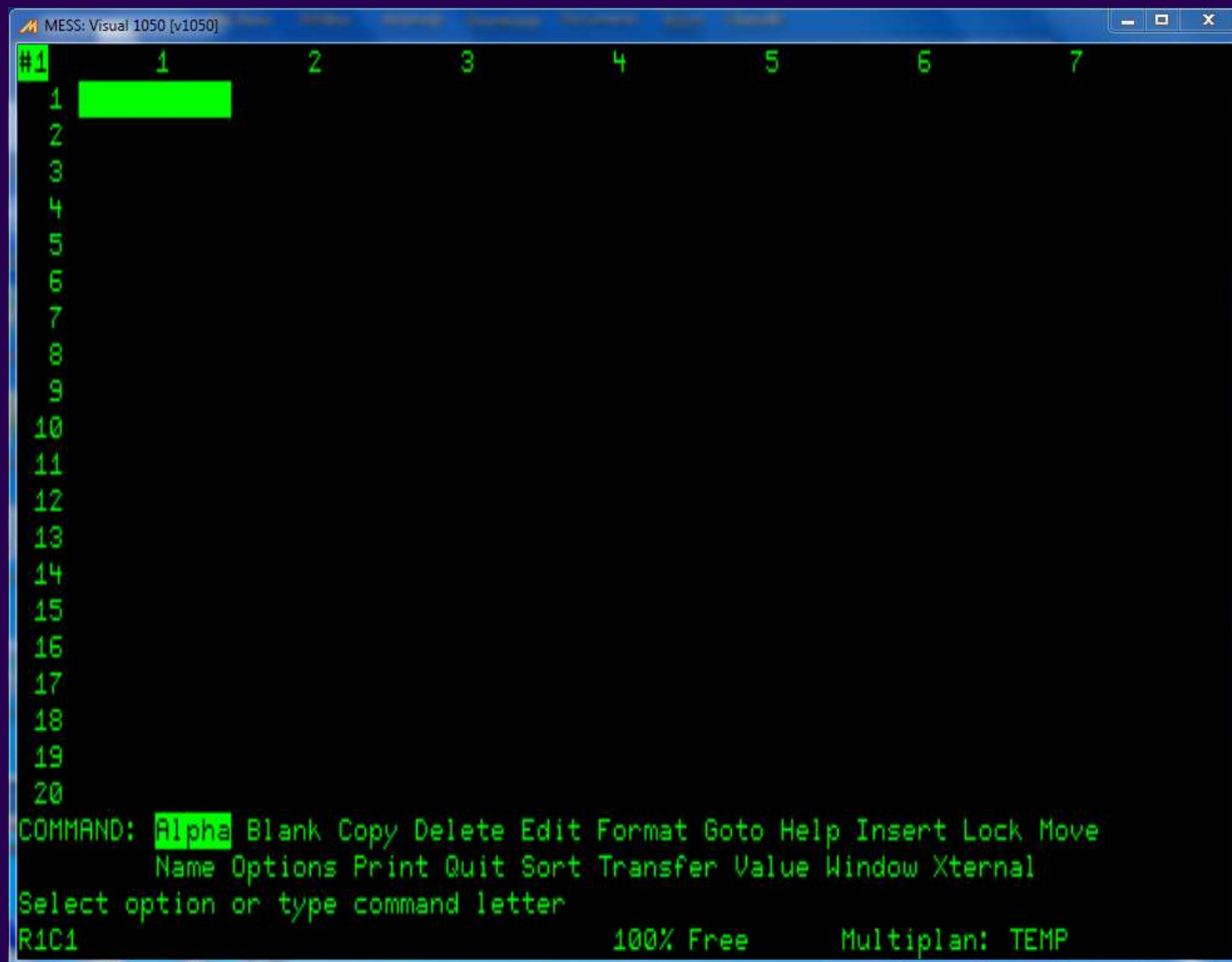
- Contratar a los *mejores desarrolladores*:  
*“Microsoft can achieve with a few hundred top-notch developers that for which IBM would need thousands.”*
- Darles *libertad*
- Los equipos para los diferentes productos son en gran medida *independientes*
- Ciclos de desarrollo relativamente *cortos*:
  - Actualizaciones de versión cada 1 o 2 meses
  - Nuevos productos cada 1 a 4 años
  - Conducido por fecha de release
- Poca especificación: permite *flexibilidad* para hacer cambios (tales como modificación de funcionalidad entregada).

# Primeras épocas (circa 1984)

## *Separación* de testing y desarrollo:

- Quejas de fabricantes de *hardware* (por ejemplo, por cálculos erróneos en BASIC).
- Quejas de clientes acerca de los *productos*.
- IBM insistió en que MS mejore el proceso de desarrollo y *control de calidad*.
- Un bug serio con consecuencias de destrucción de datos forzó a que MS desarrolle y distribuya una actualización de Multiplan a unos 20.000 usuarios con un costo de 10 USD c/u.
- En dinero de hoy, USD 200.000 equivale a unos USD 486.500.
- En 1984 MS facturó unos USD 97.479.000.

# Multiplan (circa 1984)



# Primeras épocas (circa 1984)

## *Separación* de testing y desarrollo (cont.):

- Resistencia de desarrolladores y algunos gerentes (incluyendo a Steve Ballmer, asistente al presidente y CEO 2000–2014):

*“Developers could test their own products, assisted on occasion by high school students, secretaries, and some outside contractors.”*

- Contrataron *testers externos*:
  - Grupo de testing separado
  - Tests automatizados
  - Revisiones de código para gente nueva y componentes críticos
- Evitaron así la burocracia de las *inspecciones formales*, aprobación para pasar de etapas, bitácoras de actividades, etc.



# Primeras épocas (circa 1986)

## *Grupos* de testing:

- Los desarrolladores “se pusieron holgazanes” y empezaron a contar con las actividades del equipo de test para lograr QA.
- “*Defectos infinitos*”: Los testers encuentran defectos más rápido de lo que los desarrolladores pueden arreglar.
- Integraciones tardías y grandes (“*big bang*”): períodos de testing largos, entregas tardías.
- *Desastre de Mac Word 3*: 8 meses de demora, cientos de bugs (incluyendo *crashes* y bugs con destrucción de datos).
- USD 1.000.000 en actualizaciones gratuitas.
- Creció la presión de *entregar calidad*.

# El retiro de 1989 y “Cero defectos”

## **Microsoft Memo**

To: Application developers and testers

From: Chris Mason

Date: 6/20/89

Subject: Zero-defects code

Cc: Mike Maples, Steve Ballmer, Applications Business Unit managers and department heads

---

“On May 12th and 13th, the applications development managers held a retreat with some of their project leads, Mike Maples, and other representatives of Applications and Languages. My discussion group investigated techniques for writing code with no defects. This memo describes the conclusions which we reached.... *There are a lot of reasons why our products seem to get buggier and buggier. It's a fact that they're getting more complex, but we haven't changed our methods to respond to that complexity....* The point of enumerating our problems is to realize that our current methods, not our people, cause their own failure.... Our scheduling methods and Microsoft's culture encourage doing the minimum work necessary on a feature. When it works well enough to demonstrate, we consider it done, everyone else considers it done, and the feature is checked off the schedule. The inevitable bugs months later are seen as unrelated.... When the schedule is jeopardized, we start cutting corners.... *The reason that complexity breeds bugs is that we don't understand how the pieces will work together.* This is true for new products as well as for changes to existing products.... I mean this literally: your goal should be to have a working, nearly-shippable product every day.... Since human beings themselves are not fully debugged yet, there will be bugs in your code no matter what you do. When this happens, you must evaluate the problem and resolve it immediately.... Coding is the major way we spend our time. Writing bugs means we're failing in our major activity. *Hundreds of thousands of individuals and companies rely on our products; bugs can cause a lot of lost time and money. We could conceivably put a company out of business with a bug in a spreadsheet, database, or word processor. We have to start taking this more seriously*” (italics added) (Mason, 1989).



# El retiro de 1989 y “Cero defectos”

## **Microsoft Memo**

To: Application developers and testers

From: Chris Mason

Date: 6/20/89

Subject: Zero-defects code

Cc: Mike Maples, Steve Ballmer, Applications Business Unit managers and department heads

---

“On May 12th and 13th, the applications development managers held a retreat with some of their project leads, Mike Maples, and other representatives of Applications and Languages. My discussion group investigated techniques for writing code with no defects. This memo describes the conclusions which we reached.... *There are a lot of reasons why our products seem to get buggier and buggier. It's a fact that they're getting more complex, but we haven't changed our methods to respond to that complexity.... The point of enumerating our problems is to realize that our current methods, not our people, cause their own failure....* Our scheduling methods and Microsoft's culture encourage doing the minimum work necessary on a feature. When it works well enough to demonstrate, we consider it done, everyone else considers it done, and the feature is checked off the schedule. The inevitable bugs months later are seen as unrelated.... When the schedule is jeopardized, we start cutting corners.... *The reason that complexity breeds bugs is that we don't understand how the pieces will work together.* This is true for new products as well as for changes to existing products.... I mean this literally: your goal should be to have a working, nearly-shippable product every day.... Since human beings themselves are not fully debugged yet, there will be bugs in your code no matter what you do. When this happens, you must evaluate the problem and resolve it immediately.... Coding is the major way we spend our time. Writing bugs means we're failing in our major activity. *Hundreds of thousands of individuals and companies rely on our products; bugs can cause a lot of lost time and money. We could conceivably put a company out of business with a bug in a spreadsheet, database, or word processor. We have to start taking this more seriously*” (italics added) (Mason, 1989).

# El retiro de 1989 y “Cero defectos”

## **Microsoft Memo**

To: Application developers and testers

From: Chris Mason

Date: 6/20/89

Subject: Zero-defects code

Cc: Mike Maples, Steve Ballmer, Applications Business Unit managers and department heads

---

“On May 12th and 13th, the applications development managers held a retreat with some of their project leads, Mike Maples, and other representatives of Applications and Languages. My discussion group investigated techniques for writing code with no defects. This memo describes the conclusions which we reached.... *There are a lot of reasons why our products seem to get buggier and buggier. It's a fact that they're getting more complex, but we haven't changed our methods to respond to that complexity....* The point of enumerating our problems is to realize that our current methods, not our people, cause their own failure.... Our scheduling methods and Microsoft's culture encourage doing the minimum work necessary on a feature. When it works well enough to demonstrate, we consider it done, everyone else considers it done, and the feature is checked off the schedule. The inevitable bugs months later are seen as unrelated.... When the schedule is jeopardized, we start cutting corners.... *The reason that complexity breeds bugs is that we don't understand how the pieces will work together.* This is true for new products as well as for changes to existing products.... I mean this literally: your goal should be to have a working, nearly-shippable product every day.... Since human beings themselves are not fully debugged yet, there will be bugs in your code no matter what you do. When this happens, you must evaluate the problem and resolve it immediately.... Coding is the major way we spend our time. Writing bugs means we're failing in our major activity. *Hundreds of thousands of individuals and companies rely on our products; bugs can cause a lot of lost time and money. We could conceivably put a company out of business with a bug in a spreadsheet, database, or word processor. We have to start taking this more seriously*” (italics added) (Mason, 1989).



# El retiro de 1989 y “Cero defectos”

## **Microsoft Memo**

To: Application developers and testers

From: Chris Mason

Date: 6/20/89

Subject: Zero-defects code

Cc: Mike Maples, Steve Ballmer, Applications Business Unit managers and department heads

---

“On May 12th and 13th, the applications development managers held a retreat with some of their project leads, Mike Maples, and other representatives of Applications and Languages. My discussion group investigated techniques for writing code with no defects. This memo describes the conclusions which we reached.... *There are a lot of reasons why our products seem to get buggier and buggier. It's a fact that they're getting more complex, but we haven't changed our methods to respond to that complexity....* The point of enumerating our problems is to realize that our current methods, not our people, cause their own failure.... Our scheduling methods and Microsoft's culture encourage doing the minimum work necessary on a feature. When it works well enough to demonstrate, we consider it done, everyone else considers it done, and the feature is checked off the schedule. The inevitable bugs months later are seen as unrelated.... When the schedule is jeopardized, we start cutting corners.... *The reason that complexity breeds bugs is that we don't understand how the pieces will work together.* This is true for new products as well as for changes to existing products.... I mean this literally: your goal should be to have a working, nearly-shippable product every day.... Since human beings themselves are not fully debugged yet, there will be bugs in your code no matter what you do. When this happens, you must evaluate the problem and resolve it immediately.... Coding is the major way we spend our time. Writing bugs means we're failing in our major activity. *Hundreds of thousands of individuals and companies rely on our products; bugs can cause a lot of lost time and money. We could conceivably put a company out of business with a bug in a spreadsheet, database, or word processor. We have to start taking this more seriously*” (italics added) (Mason, 1989).

# Reglas de cero defectos para Excel 4

Excel 4 fue entregado en 1992:

- Todos los *cambios* debían:
  - compilar y “linkear” (enlazar adecuadamente)
  - pasar las pruebas rápidas tanto en Mac como en Windows
- Todo desarrollador que tenga más de 10 bugs abiertos asignados debe arreglarlos antes de trabajar en nuevas funcionalidades.

# Testing buddies

- Separación de equipos de testing y de desarrollo, aproximadamente del *mismo tamaño*.
- Los desarrolladores testean su propio código y corren tests automatizados *diariamente*.
- A menudo se asignaban testers *individuales* a un desarrollador para:
  - probar sus releases privadas (*branches*)
  - darle *devoluciones rápidas* por correo electrónico antes de que se hiciera el *merge* del código.

# Testers

- Eran motivados a *comunicarse* con el equipo de soporte y los clientes, y hacer revisiones de evaluaciones.
- Desarrollaban estrategias de testing para áreas de *alto riesgo*.
- Muchas *formas* de testing, internamente llamadas:
  - Testing no estructurado
  - Testing ad hoc
  - Testing *gorila* (un tipo de testing exhaustivo y repetitivo, comparar con testing “mono”)
  - Viernes de “estilo libre” (*Free-form Fridays*)

# Primera parte de los 1990s

- Meta de *cero defectos* (explicitada en el memo de 1989).
- *Hitos* (comenzaron con MS Publisher 1.0 en 1988).
- Control de versiones, branches, integración frecuente.
- Builds diarios.
- Tests automatizados (“*quick autotest*”) que deben tener éxito *antes* de hacer check-in.
- Laboratorios de *usabilidad*.

# Primera parte de los 1990s

- Beta testing con *instrumentación* de código: 400.000 beta testers para Windows 95.
- Revisiones *formales* breves de diseño; revisiones de código para secciones *particulares*.
- Rastreo de *defectos* y *métricas*.
- Los desarrolladores se quedaban en el grupo del producto por *más de un ciclo* de entrega.



# Métricas

- Severidad:
  - 1: *Crash* del producto
  - 2: *Crash* de una funcionalidad
  - 3: Bug con arreglo posible ya identificado
  - 4: Cosmético/menor
- Cantidad de bugs *abiertos*, por severidad:
  - La cantidad de bugs abiertos debería *disminuir* antes de un hito.
  - Todos los bugs *severos* conocidos deben ser corregidos antes de entregar.
  - Se mantienen *datos de métricas* a través de diferentes entregas y proyectos.

# Métricas

- Métricas de *performance*
- Se usa la información para decidir cuándo se está “listo para entregar”:
  - Relativo y *pragmático*, no una visión absolutista
  - “*El mercado perdonará que entreguemos tarde, pero no perdonará los bugs.*”

# Desafíos en la cultura de Microsoft

- Poca comunicación entre equipos de *diferentes productos*.
- Los desarrolladores y testers muchas veces tenían poco entrenamiento o educación en *ingeniería de software*.
- Por ende, reinventaban la rueda; por ejemplo al *subestimar* conceptos y prácticas tales como:
  - Arquitectura
  - Diseño
  - Compartir componentes
  - Métricas de calidad
- Los desarrolladores se *resistían* al cambio y a la “burocracia”.

# “*Post-mortems*” de proyectos

- Objetivo: Identificar *problemas sistemáticos* y *buenas prácticas* (reporte de entre 10 y 150 páginas):
  - Hitos probados
  - Especificación insuficiente
  - No hacer revisión de commits
  - Utilizar lenguaje de aserción para comunicar suposiciones
  - Falta de herramientas adecuadas en tests automatizados
  - Versiones de código instrumentado para testers y entregas beta
  - La regla de cero defectos no es una prioridad para los desarrolladores
- Las lecciones aprendidas se hacían circular vía *memos* para motivar el *aprendizaje entre equipos*.

# Auditorías de proceso

- *Auditorías informales* de una semana focalizadas en áreas problemáticas.
- Análisis de *métricas*.
- *Entrevistas* a miembros de equipos.
- *Recomendaciones* para adoptar buenas prácticas de otros equipos, tales como:
  - Builds diarios
  - Tests automatizados
  - Hitos
  - Revisiones

# Revisiones de código

- Herramientas de *revisión de código* propias, compartidas entre equipos.
- Estudios internos acerca de la *efectividad* de las revisiones de código.
- Herramientas internas para *mejorar* las revisiones.

# A partir del año 2001: SLAM/SDV

- Proyecto desarrollado en *Microsoft Research*.
- Implementado en lenguaje funcional “OCaml” utilizando tecnología de *model checking* en código C.
- Objetivo: Reducir la cantidad de *pantallas azules*, las cuales en general eran causadas por problemas con los *drivers* (desarrollados por terceros).
- Encuentra clases particulares de *violaciones de protocolo*:
  - Utilizando características de los drivers (no código C general).
  - Encontró varios bugs en drivers supuestamente “bien testeados”.



# A partir del año 2001: SLAM/SDV

- Se incorporó a la *suite de compiladores* de Microsoft en forma totalmente *automatizada*.
- Disponible en forma gratuita.
- Parte del programa de *certificación de drivers*.
- Fue un *éxito* desde el punto de vista del negocio: eliminó la mayoría de las pantallas azules.
- También fue un éxito desde el punto de vista de la *investigación básica*: las herramientas se originaron en laboratorios universitarios con *financiamiento público*.





# El memo de 2002

- En 2002, Bill Gates publicó un memo titulado “*Trustworthy Computing*”, definiendo este término en función de:
  - Disponibilidad
  - Confiabilidad
  - Seguridad (con énfasis por encima de funcionalidad)
- Gates compara estas metas con lo que comúnmente se espera de los servicios de electricidad, agua y teléfono.
- También identifica a los sistemas de información como “*partes integrales e indispensables de casi todo lo que haremos de acá a 10 años*”.

# El memo de 2002

- En 2002, Bill Gates publicó un memo titulado “*Trustworthy Computing*”.
- Lo acompaña un *white paper*:



Se encuentra en el repositorio de la materia.

Accesible también en:

<http://news.microsoft.com/2012/01/11/memo-from-bill-gates/>

[http://download.microsoft.com/documents/australia/about/trustworthy\\_comp.doc](http://download.microsoft.com/documents/australia/about/trustworthy_comp.doc)

# A partir de 2010: Ágil

- Los servicios basados en Web y la evolución de C++ llevaron a la necesidad de *iterar más rápidamente*.
- Se adoptaron las metodologías *ágiles*.
- *Reducción masiva* del equipo de testing:
  - Se pasó de alrededor de dos testers por desarrollador a sólo uno.
  - Pero, se espera que los desarrolladores también hagan testing.

# QA en los procesos de desarrollo de software: *Discusión introductoria*

# Algunos puntos importantes

- Pedir *entregables* de QA como parte de los *hitos*: políticas gerenciales de pasar inspecciones o entregar reportes antes de dar por completo un hito.
- Cambiar prácticas de desarrollo (esto requiere *colaboración* por parte de los desarrolladores):
  - Integración continua
  - Pair programming
  - Check-ins revisados
  - Pasar pruebas de análisis estático con *zero bugs* antes de hacer check-in.

# Algunos puntos importantes

- Mantener información acerca de bugs y otras *métricas* de calidad.
- Forzar que pruebas basadas en análisis estático sean *parte de las revisiones* de código.
- Google adoptó esta práctica, veamos un ejemplo:



# Análisis estático en Google

```
package com.google.devtools.staticanalysis;
```

```
public class Test {
```

▼ Lint Missing a Javadoc comment.

Java

1:02 AM, Aug 21

[Please fix](#)

[Not useful](#)

```
public boolean foo() {  
    return getString() == "foo".toString();  
}
```

▼ ErrorProne String comparison using reference equality instead of value equality  
(see <http://code.google.com/p/error-prone/wiki/StringEquality>)

StringEquality

1:03 AM, Aug 21

[Please fix](#)

Suggested fix attached: [show](#)

[Not useful](#)

```
}  
  
public String getString() {  
    return new String("foo");  
}  
}
```

# Análisis estático en Google

```
package com.google.devtools.staticanalysis;
```

```
public class Test {
```

▼ Lint

Missing a Javadoc comment.

Java

1:02 AM, Aug 21

[Please fix](#)

[Not useful](#)

```
public boolean foo() {  
    return getString() == "foo".toString();  
}
```

▼ ErrorProne

String comparison using reference equality instead of value equality  
(see <http://code.google.com/p/error-prone/wiki/StringEquality>)

StringEquality

1:03 AM, Aug 21

[Please fix](#)

//depot/google3/java/com/google/devtools/staticanalysis/Test.java

```
package com.google.devtools.staticanalysis;
```

```
public class Test {  
    public boolean foo() {  
        return getString() == "foo".toString();  
    }  
}
```

```
    public String getString() {  
        return new String("foo");  
    }  
}
```

```
package com.google.devtools.staticanalysis;
```

```
import java.util.Objects;
```

```
public class Test {  
    public boolean foo() {  
        return Objects.equals(getString(), "foo".toString());  
    }  
}
```

```
    public String getString() {  
        return new String("foo");  
    }  
}
```

Apply

Cancel

# Rastreo de defectos/asuntos

- “*Issues*” (asuntos): Bugs, pedidos de funcionalidad, consultas, etc.
- Bases de las *métricas*:
  - Fase en la que fue reportado
  - Duración/dificultad estimada para reparar
  - Categorización (análisis de causa raíz)
- Facilita la *comunicación*
  - Se pueden realizar consultas con la persona que lo informó.
  - Se asegura que nada de lo que se reporta quede en el olvido.
- Asignación de *responsabilidad* (“*accountability*”).



# Rastreo de defectos/asuntos

The screenshot displays the HappyFox Inc. support ticketing system interface. The top navigation bar includes links for New Ticket, Forum, My Settings, Billing, John Doe, Support, and Logout. Below this, a secondary navigation bar shows Dashboard, Tickets (selected), Manage, Contacts, Reports, and Knowledge Base. A third navigation bar lists ticket statuses: My Queue, Pending (selected), All, New, Open, On Hold, SPAM, Solved, and Closed.

The main content area is titled 'Tickets' and shows 'showing 1 - 20 of 38'. It includes a search bar with the placeholder 'Search Tickets' and a 'SEARCH' button. Below the search bar, there are filters for Actions, Sort by, All Categories, and a pagination control showing '1 - 20'.

Three tickets are listed:

- Ticket #CO00001322:** Status: NEW. Subject: 'The application stops working on my Internet Explorer ? (1)'. Description: 'The application stops working on my Internet Explorer ?'. Last updated: 1 minute ago. Assigned to: james. Raised by: Jack Smith. Priority: High. Category: Customer Support. Due in: 21 days.
- Ticket #CO00001321:** Status: SOLVED. Subject: 'I cannot downgrade my account from Platinum to Gold when i'm still und ... (1)'. Description: 'I cannot downgrade my account from Platinum to Gold when i'm still under older subscription plan ?'. Last updated: 3 minutes ago. Assigned to: scott. Raised by: Abraham S. Priority: High. Category: Customer Support. Due in: 20 days.
- Ticket #CO00001320:** Status: ON HOLD. Subject: 'I cannot upgrade upload a .PNG image onto my widget ? Does your widget ... (1)'. Description: 'I cannot upgrade upload a .PNG image onto my widget ? Does your widget supports .PNG format ?'. Last updated: 5 minutes ago. Assigned to: james. Raised by: Melissa Smith. Priority: Normal. Category: Customer Support. Due in: 27 days.

# Aplicación/Conformidad

Ejemplos de cómo lograr la conformidad en la práctica:

- **Microsoft:** “Barreras” de check-in: no se puede hacer check-in sin pasar una suite de análisis **sin errores** de:
  - Cobertura de tests
  - Violaciones de dependencias
  - Mala calidad de diseño
  - Overflow de enteros
  - Aritmética de asignación
  - Accesos fuera de rango en buffers
  - Errores de memoria
  - Problemas de seguridad
  - etc.

# Aplicación/Conformidad

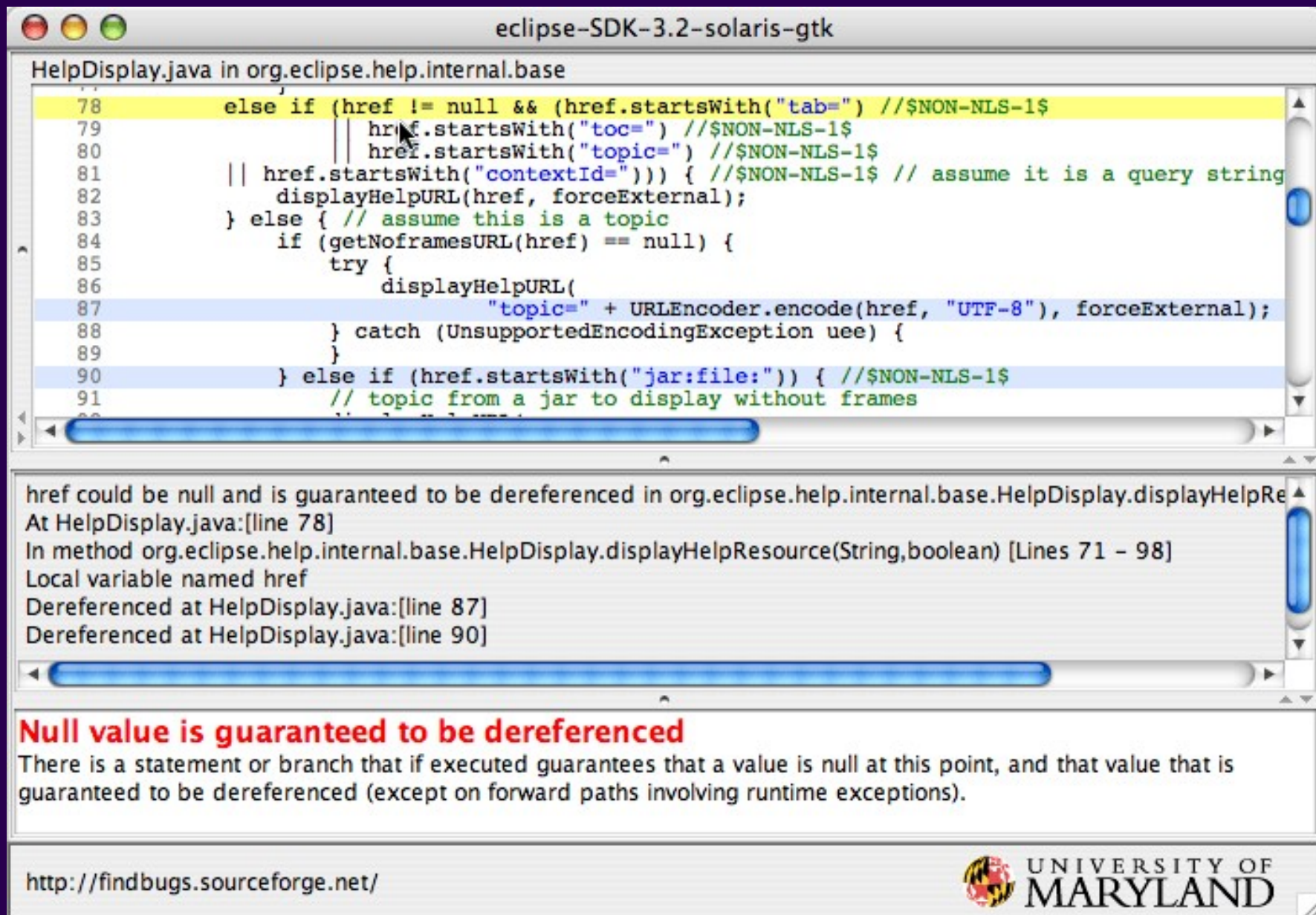
Ejemplos de cómo lograr la conformidad en la práctica (cont.):

- **Google**: Análisis estático al hacer commits, aparecen los resultados en las revisiones.
- **eBay**: pasaje de código de los desarrolladores al área de QA, ambos corren una herramienta llamada FindBugs (otro sistema desarrollado en una universidad; luego fue sucedido por SpotBugs).





# FindBugs encuentra bugs simples...



The screenshot shows the Eclipse IDE window titled "eclipse-SDK-3.2-solaris-gtk". The editor displays the file "HelpDisplay.java" in the package "org.eclipse.help.internal.base". The code is as follows:

```
78 else if (href != null && (href.startsWith("tab=") //$NON-NLS-1$
79         || href.startsWith("toc=") //$NON-NLS-1$
80         || href.startsWith("topic=") //$NON-NLS-1$
81         || href.startsWith("contextId="))) { //$NON-NLS-1$ // assume it is a query string
82     displayHelpURL(href, forceExternal);
83 } else { // assume this is a topic
84     if (getNoframesURL(href) == null) {
85         try {
86             displayHelpURL(
87                 "topic=" + URLEncoder.encode(href, "UTF-8"), forceExternal);
88         } catch (UnsupportedEncodingException uee) {
89             // ignore
90         } else if (href.startsWith("jar:file:")) { //$NON-NLS-1$
91             // topic from a jar to display without frames
```

Below the code editor, the FindBugs plugin has detected a bug. The message reads:

href could be null and is guaranteed to be dereferenced in org.eclipse.help.internal.base.HelpDisplay.displayHelpResource  
At HelpDisplay.java:[line 78]  
In method org.eclipse.help.internal.base.HelpDisplay.displayHelpResource(String,boolean) [Lines 71 - 98]  
Local variable named href  
Dereferenced at HelpDisplay.java:[line 87]  
Dereferenced at HelpDisplay.java:[line 90]

The bug is summarized as:

**Null value is guaranteed to be dereferenced**  
There is a statement or branch that if executed guarantees that a value is null at this point, and that value that is guaranteed to be dereferenced (except on forward paths involving runtime exceptions).

At the bottom of the window, there is a link to <http://findbugs.sourceforge.net/> and the logo of the University of Maryland.

# ...y también más difíciles

The screenshot shows the FindBugs application interface. The left pane displays a project tree with the following structure:

- edu.umd.cs.findbugs.config (3)
- edu.umd.cs.findbugs.filter (1)
- edu.umd.cs.findbugs.util (1)
  - Medium (1)
    - Bad practice (1)
      - Stream not closed on all paths (1)
        - Method may fail to close stream (1)
          - edu.umd.cs.findbugs.util.Util.getXMLType (1)
- edu.umd.cs.findbugs.visitclass (1)
- edu.umd.cs.findbugs.workflow (2)
- java.util (2)

The right pane shows the source code of `Util.java` in `edu.umd.cs.findbugs.util`. The code is as follows:

```
97     assert true;
98     }
99     }
100     static final Pattern tag = Pattern.compile("^\\s*<([\\w+])"
101     public static String getXMLType(InputStream in) throws IO
102         if (!in.markSupported())
103             throw new IllegalArgumentException("Input stream
104
105         in.mark(5000);
106         BufferedReader r = null;
107         try {
108             r = new BufferedReader(Util.getReader(in), 2000);
109
110         String s;
111         int count = 0;
112         while (count < 4) {
113             s = r.readLine();
114             if (s == null)
115                 break;
116             Matcher m = tag.matcher(s);
117             if (m.find())
```

The bug report at the bottom of the window is titled "Method may fail to close stream" and provides the following details:

- edu.umd.cs.findbugs.util.Util.getXMLType(InputStream) may fail to close stream
- At Util.java:[line 108]
- In method edu.umd.cs.findbugs.util.Util.getXMLType(InputStream) [Lines 102 - 123]
- Need to close java.io.Reader

The description of the bug states: "The method creates an IO stream object, does not assign it to any fields, pass it to other methods that might close it, or return it, and does not appear to close the stream on all paths out of the method. This may result in a file descriptor leak. It is generally a good idea to use a finally block to ensure that streams are closed."

The footer of the window includes the URL <http://findbugs.sourceforge.net/> and the University of Maryland logo.

# Aplicación/Conformidad

- Para tener éxito se debe tener:
  - Pocos *falsos positivos*
  - Formas de *descartar alertas* que corresponden a falsos positivos
  - Los desarrolladores deben *estar de acuerdo* con el uso de herramientas como análisis estático.

# Aplicación/Conformidad

- También son importantes los aspectos *sociales*:
  - *Actitud* de los desarrolladores hacia los defectos
  - *Educación* de los desarrolladores acerca de la seguridad
  - Uso de la “*presión de pares*” para lograr aplicar prácticas de QA (como reglas acerca de “romper el build”)
  - *Culturas* de desarrolladores vs. testers (los testers suelen ser los que dan malas noticias)
  - Asuntos vs. defectos
  - Las suites de test buenas suelen ayudar a la confianza y el *collective code ownership*.

# Reportes de defectos

- Los defectos deben ser *reproducibles*.
- Los casos de test adjuntados deben ser *simples* y *generales*.
- Se debe incluir sólo *un defecto* por reporte.
- No se debe adoptar una actitud *antagónica*:
  - Los testers suelen dar malas noticias
  - Describir el problema en lenguaje no emocional
  - No asignar culpa



# Reportes de defectos (Ejemplo *malo*)





# Reportes de defectos (Ejemplo *malo*)



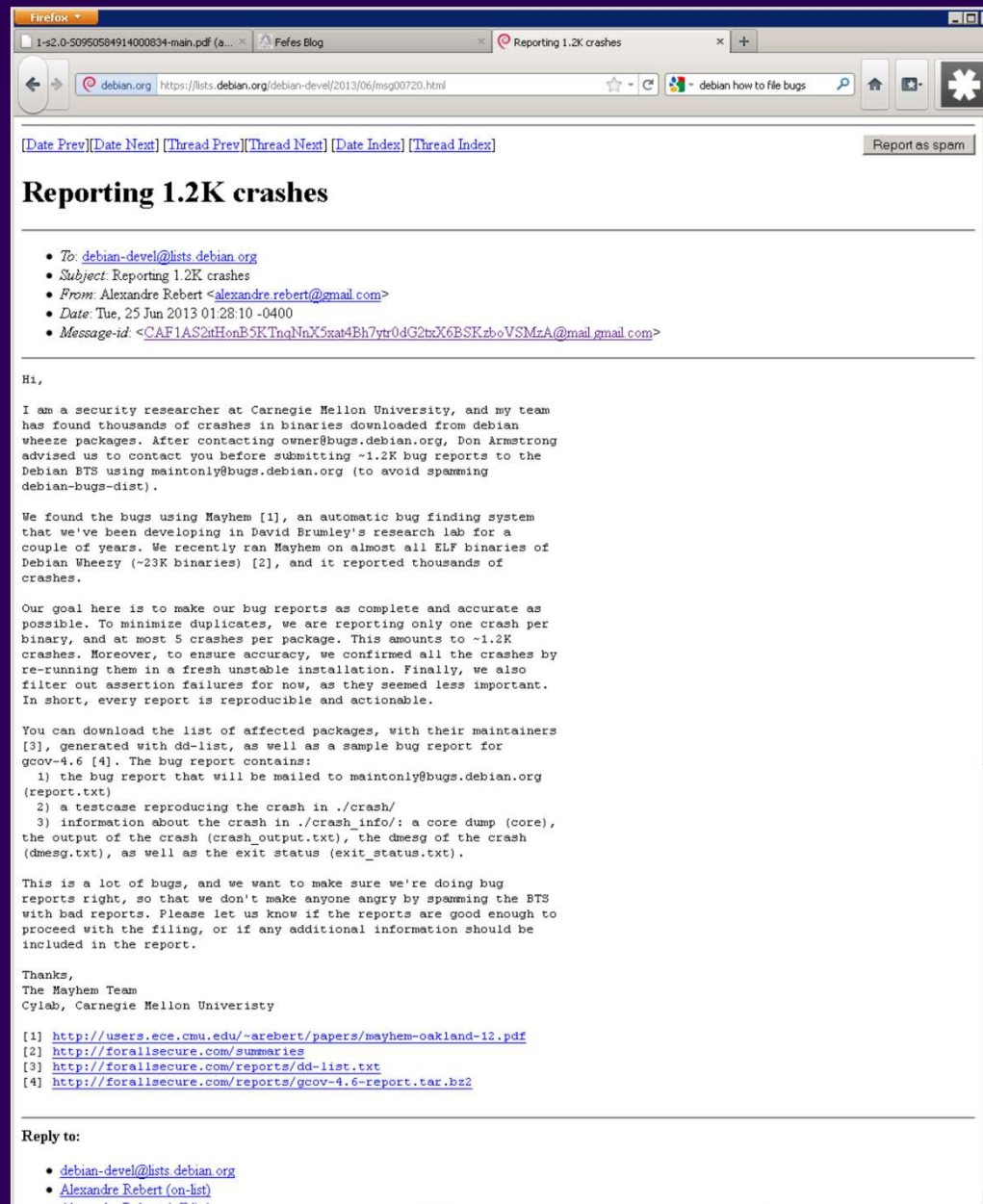
## To much is to much

I am fed up of all the bugs.  
Production never update i must go through the army to updaye the production  
disconnection very often  
loss of gold and forge point  
loss of life points of soldiers without fighting  
diasapearing soldiers  
and at las but not least my copper foundry diasapered while i was trying to change its emplacement.

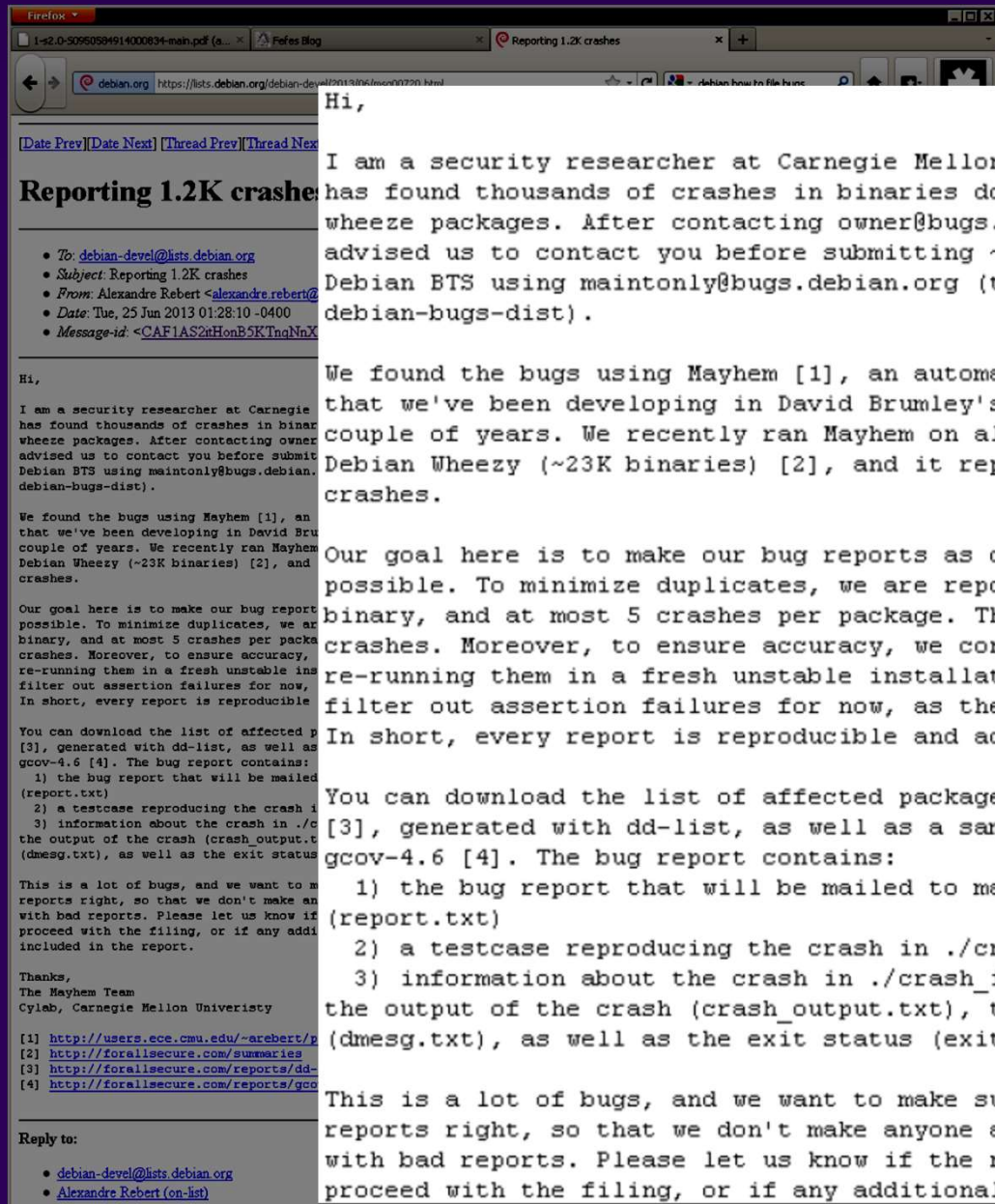
YOU ARE SORRY FOR ALL THESE INCONVNIENCE,I KNOW, YOU ARE GOING TO SAY IT IS BECAUSE IT S A BETA,I KNOW  
BUT THIS GAME SUCKS FROM THE TOP TO THE BOTTOM,PAY 10 MONKEYS AS DEVELLOPERS AND YOU WILL HAVE THE SAME RESULTS.  
BY THE WAY IF YOU WANT TO TEST A CAR BEFORE BUYING IT YOU DO NOT HAVE TO PAY,HERE WITH THE DIAMONDS OPTION IS YOU WANT TO BUY THE CAR OK, YOU WANT TO TEST IT OK SO YOU MUST PAY.  
SO PLEASE THIS TIME NO APOLOGISE,I NEED EXPLANATION AND NOT AS BETA BLA BLA BLA.  
IS INNO CIE ARE BELONGING TO BANKSTERS GANG?



# Reportes de defectos (Ejemplo *bueno*)



# Reportes de defectos (Ejemplo *bueno*)



Hi,

I am a security researcher at Carnegie Mellon University, and my team has found thousands of crashes in binaries downloaded from debian wheezy packages. After contacting owner@bugs.debian.org, Don Armstrong advised us to contact you before submitting ~1.2K bug reports to the Debian BTS using maintonly@bugs.debian.org (to avoid spamming debian-bugs-dist).

We found the bugs using Mayhem [1], an automatic bug finding system that we've been developing in David Brumley's research lab for a couple of years. We recently ran Mayhem on almost all ELF binaries of Debian Wheezy (~23K binaries) [2], and it reported thousands of crashes.

Our goal here is to make our bug reports as complete and accurate as possible. To minimize duplicates, we are reporting only one crash per binary, and at most 5 crashes per package. This amounts to ~1.2K crashes. Moreover, to ensure accuracy, we confirmed all the crashes by re-running them in a fresh unstable installation. Finally, we also filter out assertion failures for now, as they seemed less important. In short, every report is reproducible and actionable.

You can download the list of affected packages, with their maintainers [3], generated with dd-list, as well as a sample bug report for gcov-4.6 [4]. The bug report contains:

- 1) the bug report that will be mailed to maintonly@bugs.debian.org (report.txt)
- 2) a testcase reproducing the crash in ./crash/
- 3) information about the crash in ./crash\_info/: a core dump (core), the output of the crash (crash\_output.txt), the dmesg of the crash (dmesg.txt), as well as the exit status (exit\_status.txt).

This is a lot of bugs, and we want to make sure we're doing bug reports right, so that we don't make anyone angry by spamming the BTS with bad reports. Please let us know if the reports are good enough to proceed with the filing, or if any additional information should be

Thanks,  
The Mayhem Team  
Cylab, Carnegie Mellon University

[1] <http://users.ece.cmu.edu/~arebert/p>  
[2] <http://forallsecure.com/summaries>  
[3] <http://forallsecure.com/reports/dd->  
[4] <http://forallsecure.com/reports/gcov-4.6>

Reply to:

- [debian-devel@lists.debian.org](mailto:debian-devel@lists.debian.org)
- [Alexandre Rebert \(on-list\)](#)



# Referencias

Software Engineering: A Practitioner's Approach, 8th Ed. R. Pressman. McGraw-Hill, 2015.

Software Engineering, 9th Ed. I. Sommerville. Addison-Wesley, 2011.

Fundamentals of Business Process Management, 2nd Edition. M. Dumas, M. La Rosa, J. Mendling, y H.A. Reijers. Springer, 2018.

Ball, Thomas, Vladimir Levin, and SriramK. Rajamani. "A decade of software model checking with SLAM." CACM 54.7 (2011): 68-76.

Sadowski, C., van Gogh, J., Jaspan, C., Söderberg, E., & Winter, C. Tricorder: Building a Program Analysis Ecosystem. ICSE 2015.

J.L. Boria: La historia de Tahini-Tahini: Mejora de procesos de software con métodos ágiles, CreateSpace Independent Publishing Platform, 2013.

Parte del contenido de este curso está basado en:

- Material preparado por Virginia Cuomo y colegas para el dictado de cursos en UNS y UADE.
- Material incluido en el curso "Ingeniería de Software III", dictado en UNLP por Elsa Estévez y colegas.
- Material incluido en el curso "Foundations of Software Engineering", dictado en Carnegie Mellon University por C. Kästner, M. Hilton y Miguel Velez.