Testing Software

Pruebas y mantenimiento de software

Lunes de 8:00 a 10:00

en CReCE

Prof. José Antonio Cervantes Álvarez

antonio.alvarez@academicos.udg.mx

Unidad I. Fundamentos de pruebas de software

1.1 ¿Qué son las pruebas de software?

1.2 ¿Por qué son necesarias las pruebas de software?

1.3 ¿Cuál es el objetivo de las pruebas?

1.4 Conceptos básicos.

- 1.4.1 Error, defecto y fallo.
- 1.4.2 Conceptos de pruebas.
- 1.4.3 Calidad del software.
- 1.4.4 Esfuerzo de las pruebas.

1.5 El proceso fundamental de las pruebas.

- 1.5.1 Planeación y control de pruebas.
- 1.5.2 Análisis y diseño de pruebas.
- 1.5.3 Implementación y ejecución de pruebas.
- 1.5.4 Evaluación y reporte de pruebas.
- 1.5.5 Cierre de las actividades de prueba.

Unidad I. Fundamentos de pruebas de software

1.6 La psicología de las pruebas.

1.7 Principios generales de las pruebas.

1.8 La ética del ingeniero de pruebas.

¿Qué son las pruebas de software?

Antes de ver una definición de pruebas de software, veremos la percepción que tienen algunos ingenieros de software acerca de lo que son las pruebas de software:

- "Las pruebas de software son el proceso de demostrar que no hay errores presentes en un programa"
- "El propósito de las pruebas es demostrar que un programa realiza las funciones indicadas correctamente"
- "Las pruebas son el proceso de establecer confianza en que un programa hace lo que se supone debe hacer"

¿Qué son las pruebas de software?

- Jhon Myers indica que estas definiciones están mal planteadas. Cuando probamos un programa, se quiere **aportar un valor añadido** a lo que estamos probando, **elevar la calidad y fiabilidad** y esto nos lleva a tener que **encontrar los errores en el programa.**

- No tenemos que probar un programa para demostrar que funciona, sino que tenemos que partir de la suposición de que el programa va a contener errores. La definición de prueba que aporta Myers es:

"La prueba de software es el proceso de ejecución de un programa con la intención de encontrar errores"

"La prueba de software es el proceso de ejecución de un programa con la intención de encontrar errores"

- **¿Por qué se toma esta definición como valida y las anteriores no?**
  - Las pruebas no permiten demostrar que un programa esta libre de errores o fallos.
  - Si tomamos la primera definición dada ***"Las pruebas de software son el proceso de demostrar que no hay errores presentes en un programa"***, psicológicamente estaremos dirigidos hacia esa meta y tenderemos a seleccionar casos de prueba con una baja probabilidad de causar que el programa falle.
  - Si por el contrario tomamos como objetivo demostrar que un programa tiene fallos, tendremos una mayor probabilidad de encontrar los errores.

- Si tomamos la **definición "*Las pruebas son el proceso de establecer confianza en que un programa hace lo que se supone que debe hacer*"** estaríamos dando por hecho que un programa que hace lo que se supone que debe hacer, no tiene errores y eso, es un error ya que, aunque el programa realice su función, puede contener errores.

- El ISTQB (International Software Testing Qualifications Board) es una organización sin ánimos de lucro creada en el año 2002 por empresas, instituciones, organizaciones y personas especializadas en el campo de las pruebas y la industria del software. El ISTQB define las pruebas como:

"El proceso que consiste en todas las actividades del ciclo de vida, tanto estáticas como dinámicas relacionadas con la planificación, preparación y evaluación de productos de software para determinar que cumplen los requisitos especificados, para demostrar que son aptos para el propósito establecido y para detectar defectos"

- Cem Kaner, profesor de ingeniería de software en el Instituto Tecnológico de Florida. Es considerado uno de los principales gurús de las pruebas de software. Él las define como:

"Las pruebas de software son la investigación empírica y técnica realizada para facilitar a los interesados información sobre la calidad del producto o servicio bajo prueba"

Kaner introduce la figura del técnico que mediante la investigación aportará datos sobre la calidad del producto y no se centrará únicamente en la detección de errores.

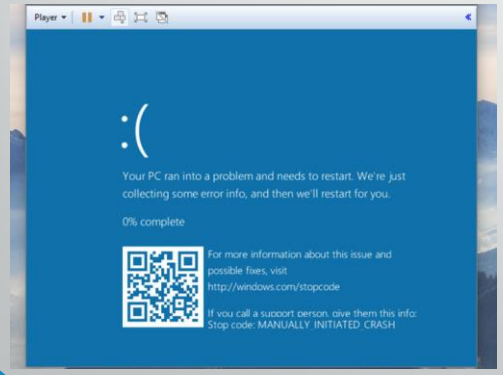
- Edsger W. Dijkstra (1930-2002) científico de la computación entre cuyas contribuciones a la ciencia es "*el algoritmo al camino más corto*". Considera que:

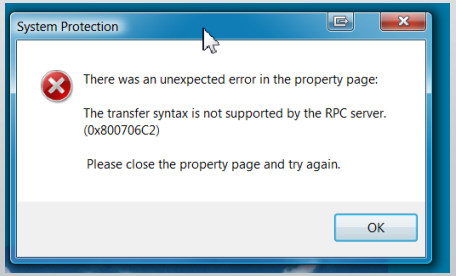
"Las pruebas de software son útiles para mostrar la presencia de fallos, pero nunca para mostrar su ausencia."

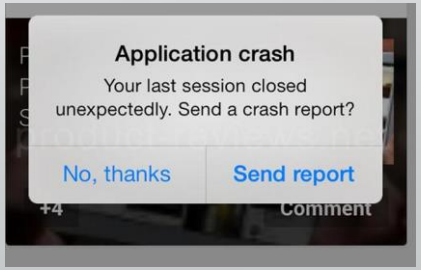
Todas y cada una de las definiciones dadas tienen algo en común. Todas ellas se centran en mayor o menor manera en la detección de errores.

¿Por qué son necesarias las pruebas de software?

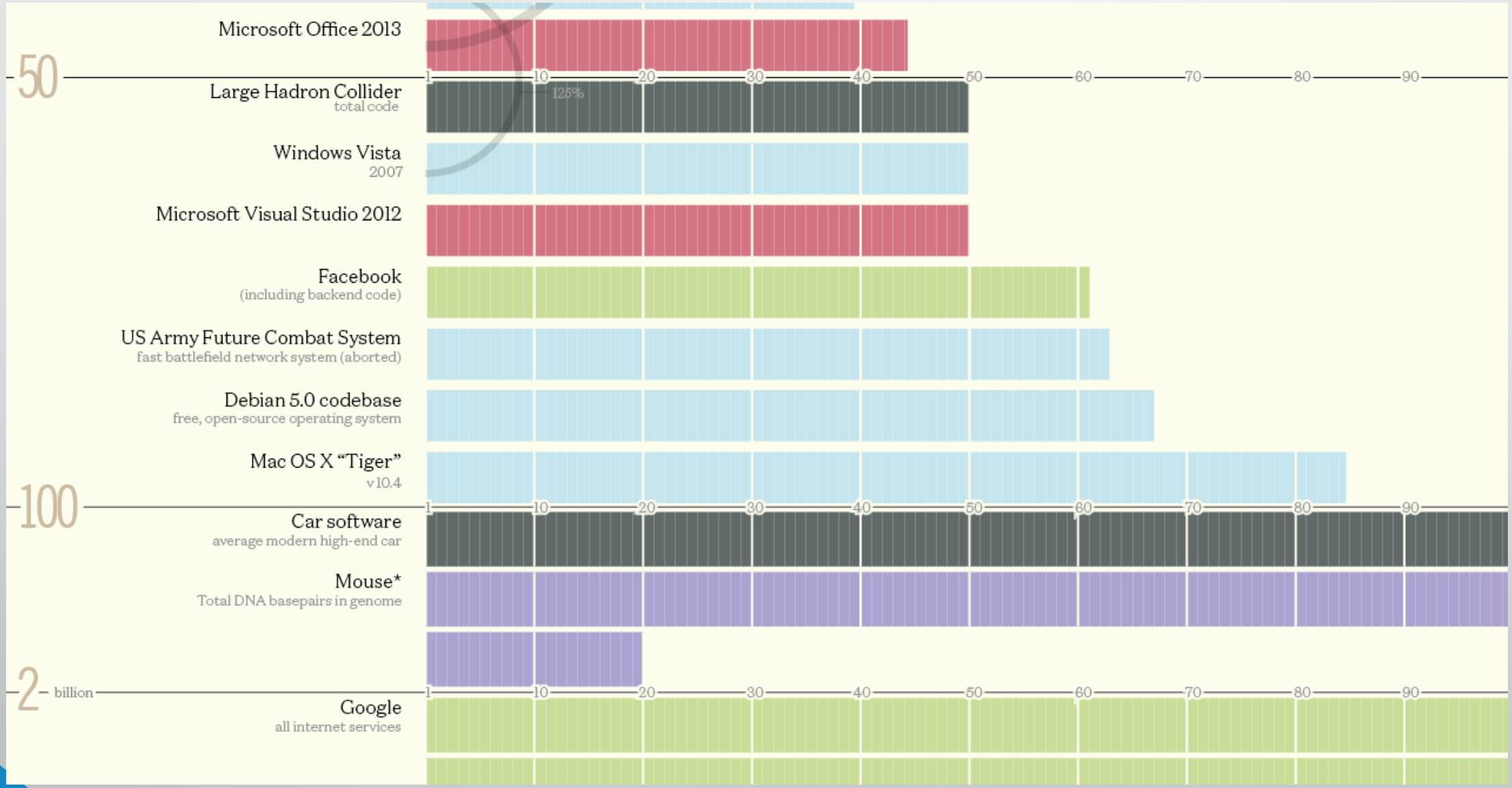
- En un proyecto de desarrollo de software los errores puede presentarse en cualquiera de las etapas del ciclo de vida del software.
- Aún cuando se intente detectarlos después de cada fase, utilizando técnicas como la inspección, algunos errores permanecen sin ser descubiertos.
- Por lo tanto es muy probable que el código final contenga **errores de requerimientos** y **diseño,** adicionales a los introducidos en la **codificación**.





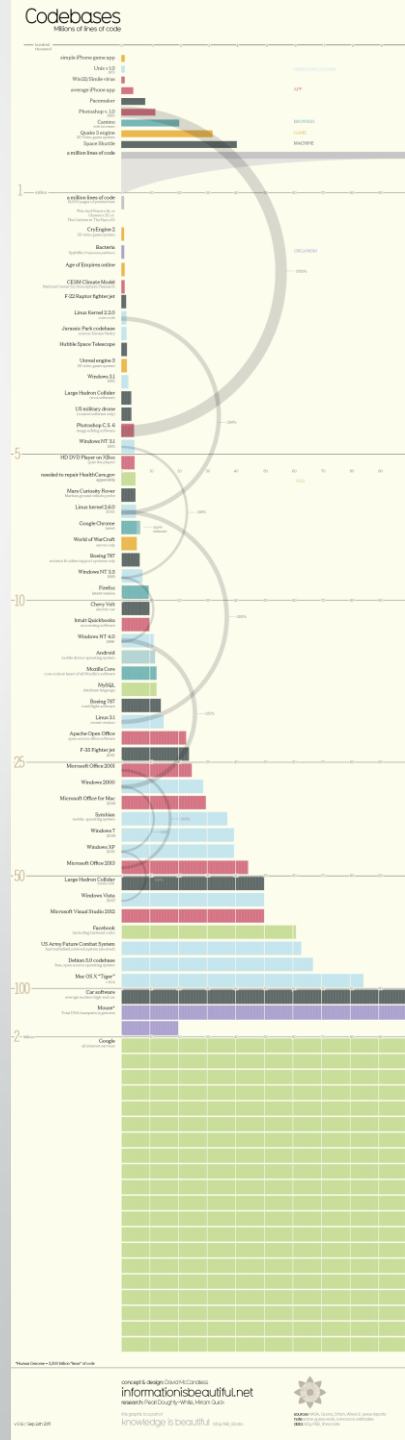


Millones de líneas de código



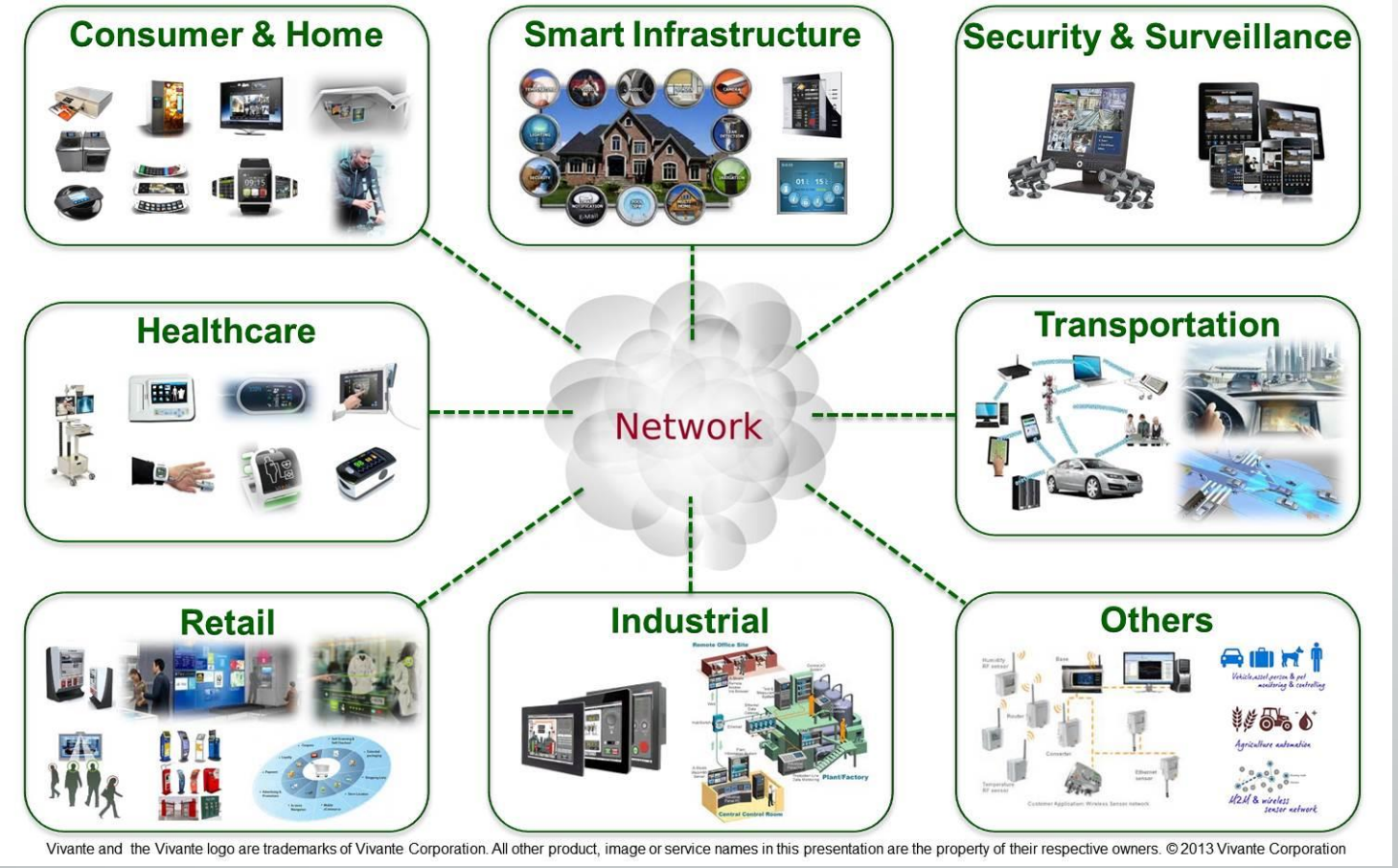
- Ejemplo del tamaño en líneas de código de algunos sistemas.

  - El software del primer transbordador especial utilizó alrededor de 400,000 líneas de código.
  - Una aplicación simple/juego para iOS puede tener entre 20,000 y 40,000 líneas de código.
  - El Sistema operative Windows 3.1 (en 1992) utilizó alrededor de 2 millones de líneas de código.
  - Un avión Boeing 787 contiene 6.5 millones de líneas de código.
  - Todo el sistema operativo Android contiene casi 12 millones de líneas de código.
  - Windows XP contiene alrededor de 40 millones de líneas de código.
  - Windows 10 está alrededor de los 80 millones de líneas de código.
  - Google y todos sus servicios como Gmail, Maps, Calendar, entre otros forman un conjunto de más de 2,000 millones de líneas de Código.



http://omicrono.elespanol.com/2016/08/lineas-de-codigo/

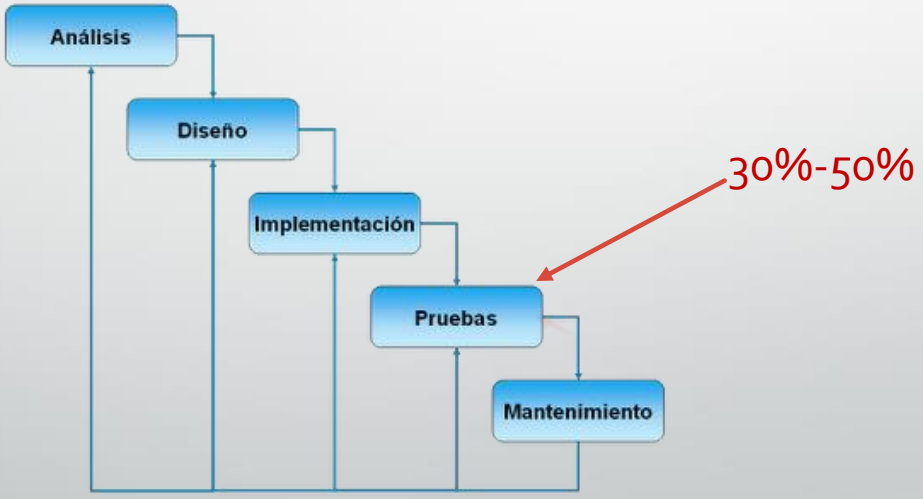
- En la actualidad vivimos en una sociedad digital, rodeados por miles de computadoras. La dependencia del ser humano hacia el software ha crecido a tal punto que no podemos imaginar como sería nuestro mundo sin él.



- Las pruebas de software son una parte importante pero muy costosa del proceso de desarrollo de software.

- Pueden llegar a representar entre el 30% y 50% del costo total del desarrollo del software.

- Sin embargo, los costos de las fallas en un software en operación pueden llegar a ser mucho mayores (catastróficos)



Para responder a la pregunta ¿por qué son necesarias las pruebas de software?, vamos a ver una serie de sucesos históricos.

Usuario demanda a LinkedIn por robo de contraseñas

Publicado el 20 junio, 2012

Una demanda colectiva apunta que la red social cometió negligencia e incumplimiento de contrato por no encriptar las bases de datos que contenían las contraseñas de sus usuarios con protocolos de seguridad estándar.

Así, la parte demandante busca una ***indemnización de 5 millones de dólares y la restitución de los passwords fugados****, luego que un hackers anunciara a principios de mes la* ***sustracción de más de 6,4 millones de contraseñas*** *y datos de usuarios de LinkedIn, lo que demuestra a su juicio "una preocupante falta de medidas de seguridad."*

- La red de AT&T se hundió en 1990 y dejó sin respuesta a 75 millones de llamadas por un fallo en su sistema de comunicaciones.

- Se colapsa el aeropuerto de Los Ángeles (2007).
- Más de 17 mil personas se quedaron en tierra por un problema de software que provocó conflictos en una tarjeta de red que bloqueó toda la red informática.

- Sobredosis radiológica en el Instituto Nacional del Cáncer de Panamá (2000)
- Errores en los procedimientos y un fallo de software causaron que se aplicaran dosis erróneas de radiación.
- 8 personas murieron y 20 tuvieron problemas de salud graves. Los médicos responsables del hecho fueron acusados de asesinato.

- Como pueden observar las pruebas de software tienen un rol muy importante en el aseguramiento de la calidad ya que permiten detectar los errores introducidos en las fases previas del proyecto.

- No probar adecuadamente el software, antes de ponerlo en producción, puede producir no solo perdidas económicas, sino también daños personales, llegando incluso a producir la muerte en algunos casos.

- Actualmente, casi todas las empresas dependen en gran medida del software, ya sea el sistema de nómina que utilizan los departamentos de finanzas de cada empresa o los equipos especializados para el desarrollo de las diversas tareas de sus empleados.

¿Cuál es el objetivo de las pruebas?

- El objetivo principal de las pruebas es aportar **calidad** al producto que se está desarrollando.

- Pero, ¿qué es calidad?

170 dólares

330 dólares

999 pesos

150 pesos

¿Cuál es el objetivo de las pruebas?

- El objetivo principal de las pruebas es aportar **calidad** al producto que se está desarrollando.

- Pero, ¿qué es calidad?

  - **ISO 9000** es un conjunto de normas sobre calidad y gestión de la calidad, la define como *"la calidad es el grado en el que un conjunto de características inherentes cumple con los requerimientos".*

  - **ISO 25000** es un conjunto de normas que tiene por objetivo la creación de un marco de trabajo común para evaluar la calidad del producto de software, la define como *"la calidad es el grado en el que el producto de software satisface las necesidades expresadas o implicadas, cuando es usado bajo condiciones determinadas".*

Conceptos básicos

- **Requerimientos.** Durante la construcción de cualquier producto, tanto las partes como el producto final son usualmente examinadas para asegurar el cumplimiento de los requerimientos dados. Estos es determinar si el producto resuelve la tarea solicitada.

- **El software es inmaterial.** Probar o evaluación un producto (parcial o final) es más complicado cuando el producto es un software.

  - No es posible examinarlo físicamente como se hace con otros productos.
  - La única forma de examinar el producto es revisar la documentación y el código.
  - Para probar la funcionalidad del software es necesario ejecutarlo para comparar su funcionalidad con los requerimientos.
  - Las pruebas y su documentación son frecuentemente definidas en el contrato o por el estándar utilizado.

Conceptos básicos

En el desarrollo de pruebas es importante comprender la diferencia entre error, fallo y defecto. Estos conceptos están relacionados entre sí, pero tienen significados diferentes.

- **Falla (failure).** Una falla se presenta cuando un requerimiento dado no es cubierto totalmente. Es una discrepancia entre la funcionalidad actual del sistema y la funcionalidad esperada.

  Ejemplo: Una falla se presenta cuando el producto es extremadamente complejo de usar o demasiado lento aún cuando cumple con los requerimientos funcionales.

- El termino falla es utilizado en el área de ingeniería de software para describir cualquier evento que permite que el usuario experimente un problema con el software.

- Las fallas son visibles cuando el programa genera un resultado erróneo o truena (an output is wrong or the program crashes)

- En el área de software las fallos ocurren como consecuencia de los ***defectos*** o ***bugs*** en el programa.

Conceptos básicos

- **Defectos (faults/bugs).** En el software los defectos están presentes desde que el software es desarrollado o modificado, pero sólo pueden hacerse visibles cuando el software es ejecutado.

- *Los defectos pueden provenir de declaraciones incorrectas (**errores** de programación) de distintas sentencias u omisiones de las mismas.*
  - *Olvidar colocar la sentencia para liberar memoria después de utilizarla.*
  - *Omitir la sentencia default en un switch.*
  - *Usar el operador relacional > en lugar de >=.*
  - *Vulnerabilidades presentes en los mecanismos de seguridad.*
  - *Etc.*

Las actualizaciones de Windows o cualquier otro software, buscan corregir los defectos encontrados después de haber lanzado el producto al mercado

- La causa de un defecto son los ***errores*** cometidos por una persona (desarrollador). Sin embargo, también pueden ser causados por condiciones del entorno como campos magnéticos, hardware dañado, etc.

Conceptos básicos

- **Error (error/mistake)**. Los programadores pueden cometer errores, principalmente cuando se trabajo bajo presión para cumplir con los tiempos de entrega.

- Olvidar implementar un requerimiento conduce a generar un software defectuoso.

- Otro motivo por el cual se generan errores es cuando el programador modifica parte de un programa sin comprender las consecuencias o alcances que pueden generar dichos cambios.

Conceptos básicos

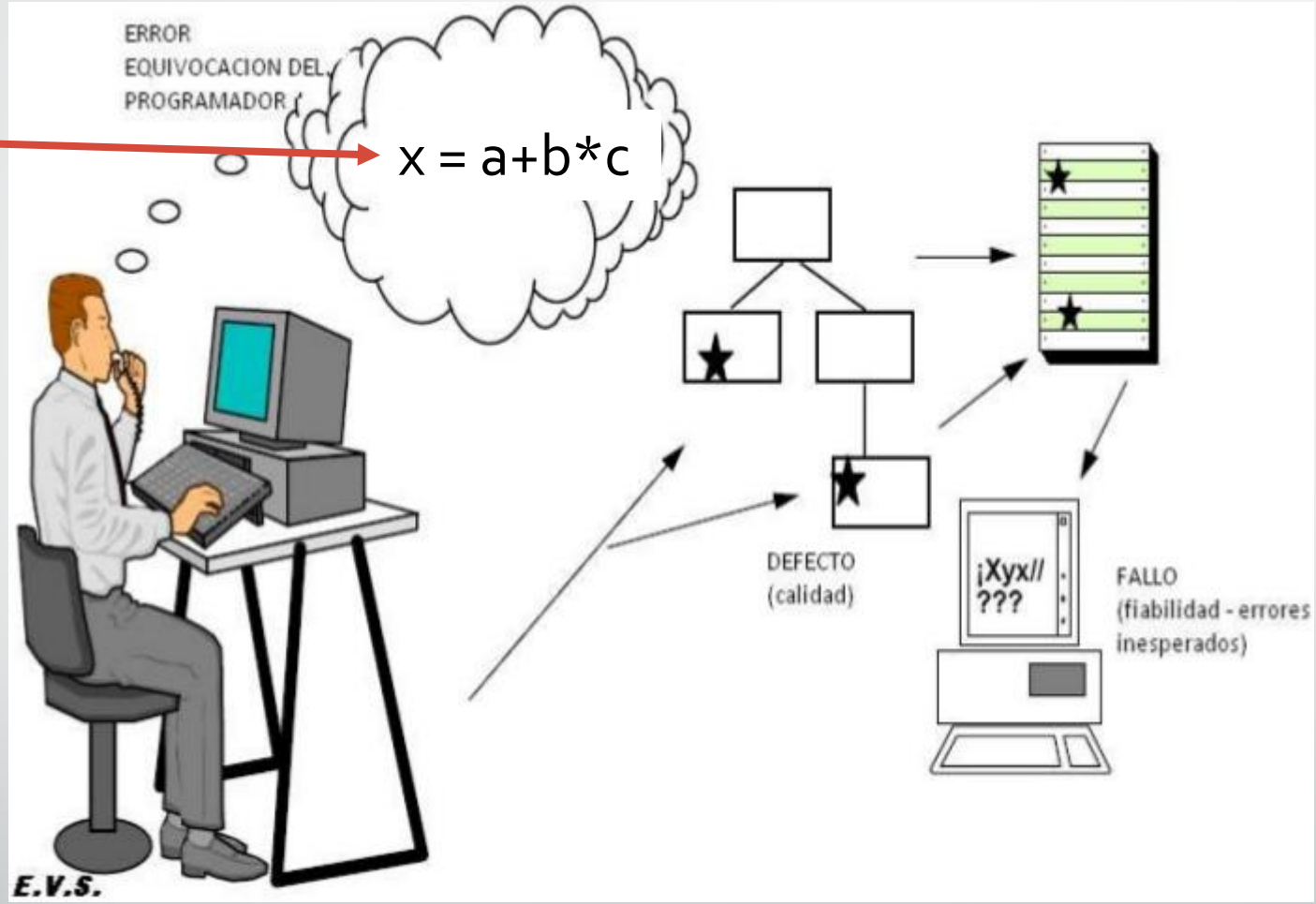
x = (a+b)*c

Ejemplo:

a = 2

b = 3

c = 5



Errores del programador → (se plasman como) Defectos en el software → (da lugar a fallas) El Sistema no se comporta como debería

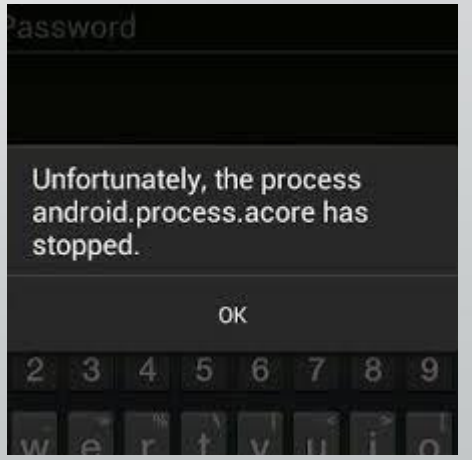
Error

Bugs

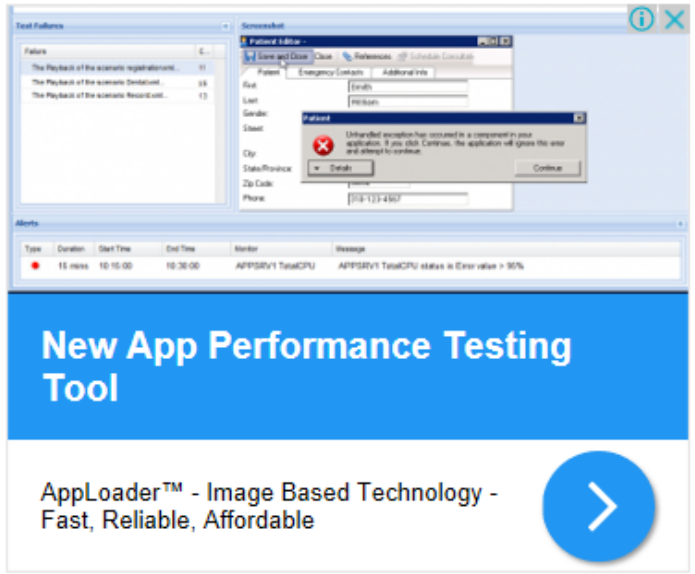
Fallas

Corregir errores

Mejorar la calidad del programa



Ejemplo de cómo reportar bugs



Bug Name: Application crash on clicking the SAVE button while creating a new user.
Bug ID: (It will be automatically created by the BUG Tracking tool once you save this bug)
Area Path: USERS menu > New Users
Build Number: Version Number 5.0.1
Severity: HIGH (High/Medium/Low) or 1
Priority: HIGH (High/Medium/Low) or 1
Assigned to: Developer-X
Reported By: Your Name
Reported On: Date
Reason: Defect
Status: New/Open/Active (Depends on the Tool you are using)
Environment: Windows 2003/SQL Server 2005

Description:
Application crash on clicking the SAVE button while creating a new user, hence unable to create a new user in the application.

Steps To Reproduce:
1) Logon into the application
2) Navigate to the Users Menu > New User
3) Filled all the user information fields
4) Clicked on 'Save' button
5) Seen an error page "ORA1090 Exception: Insert values Error..."
6) See the attached logs for more information (Attach more logs related to bug..IF any)
7) And also see the attached screenshot of the error page.

Expected result: On clicking SAVE button, should be prompted to a success message "New User has been created successfully".

CASE STUDY

Conceptos de pruebas

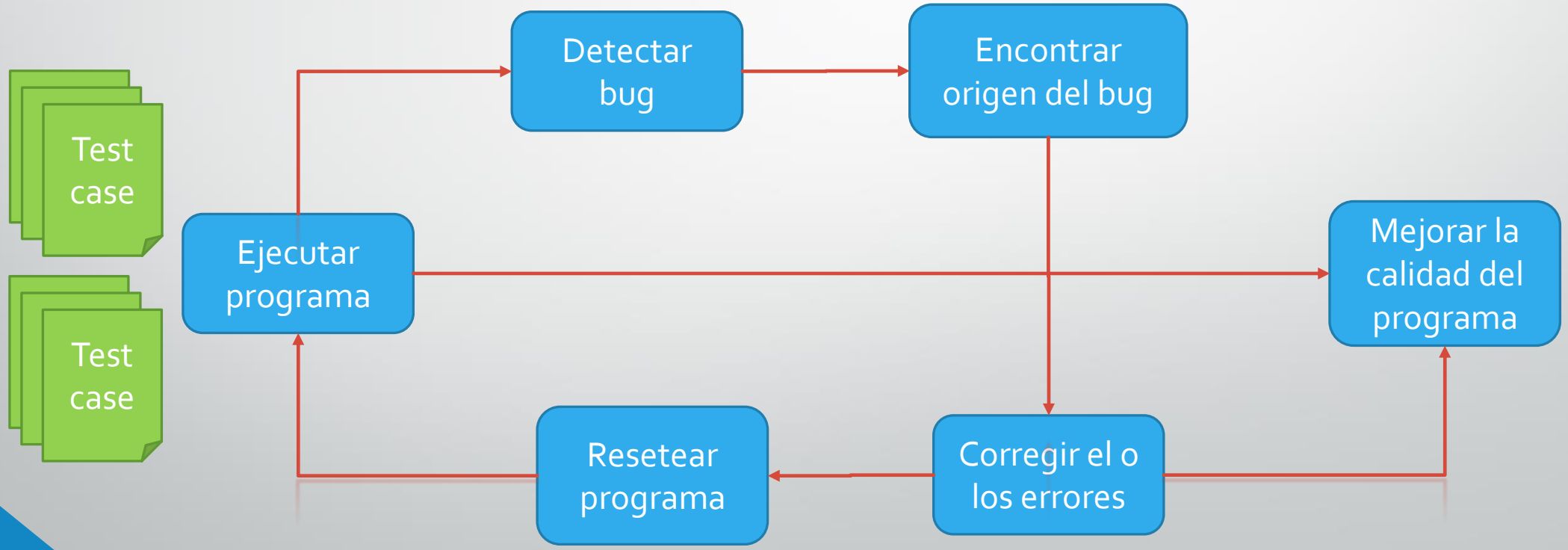
- Para poder corregir los ***bugs***, es necesario localizarlos dentro del software. Inicialmente cuando un bug es reportado se desconoce su ubicación precisa dentro del software.

- Localizar y corregir los bugs son tareas realizadas por un ingeniero de software. A este proceso generalmente se le conoce como depuración (***debugging***).

- La reparación de los ***bugs*** generalmente permite incrementar la calidad del producto.
  - Las modificaciones para corregir un bug por lo general no introduce nuevos bugs, sin embargo, en la práctica la corrección de bugs generalmente introduce nuevos bugs.
  - Los nuevos bugs introducen nuevos fallos que, por lo general se presentan con entradas totalmente diferentes a las utilizadas para descubrir el bug original.

Conceptos de pruebas

- La generación de nuevos bugs a partir de corregir bugs conocidos, hacen más complejo la tarea de depurar. Esto implica que no solo se debe repetir la ejecución de los casos de prueba utilizados para identificar el bug. Es necesario ejecutar más casos de prueba para descartar posibles bugs generados a partir de las correcciones realizadas para eliminar el bug inicial.



- **Depurar (debugging).** Es la tarea de localizar y corregir bugs. Para examinar un software es necesario el uso de pruebas que permitan identificar fallas generados por bugs.

- **Prueba (test).** Es un simple análisis o examinación de un programa.
  - Las condiciones de la prueba deben ser definidas.
  - El comportamiento generado por el software debe ser comparado con el comportamiento esperado para determinar si se satisfacen los requerimientos establecidos.

- **Probar (testing).** Es el proceso sistemático de ejecutar pruebas para demostrar la correcta implementación de los requerimientos, incrementar la confianza y detectar fallas.

- Las pruebas de software tienen diferentes propósitos:
  - Ejecutar un programa para encontrar fallas.
  - Ejecutar un programa para medir su calidad.
  - Ejecutar un programa para brindar confianza.
  - Analizar un programa o su documentación para prevenir fallas.

- **Gestor de pruebas (test management).** Utilizado para gestionar:
  - Los datos de la prueba.
  - Planeación.
  - Diseño.
  - Implementación.
  - Análisis de la prueba.

- **Repositorio de pruebas (test suite).** Esta incluye la ejecución de uno o más casos de prueba.

- **Caso de prueba (test case).** Incluye las condiciones de prueba definidas.
  - Precondiciones para la ejecución.
  - Entradas.
  - Salida esperada.

Herramientas de código abierto

1. Selenium (Web Application Testing)
2. Appium (Mobile Testing)
3. JMeter (Load Testing)
4. Jenkins (Continuous Testing)
5. TestLink (Test Management)
6. Mantis (Bug-Tracking & Project Management)
7. Postman (API Testing)
8. Firebug / Firepath (Online Debugging)
9. GitHub (Project & Source Code Hosting)
10. Bugzilla (Defect Tracking & Collaboration)
11. RazorSQL (Database Query Tool)
12. PhantomJS (Headless Browser)
13. UIAutomator (Android Testing Framework)
14. Notepad++ (Source code Editor)
15. FileZilla (FTP Solution)
16. AutoIT (Language Automation)

Conceptos básicos

https://www.youtube.com/watch?v=TEf8vJMyhsM

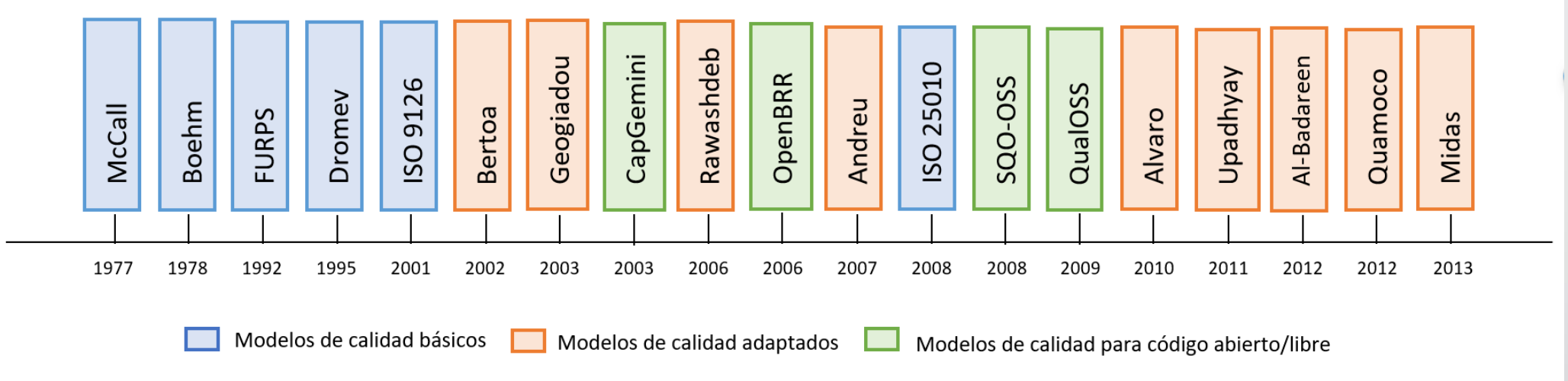
Calidad del software

- Probar el software contribuye a mejorar su calidad. Esto se realiza a través de identificar y corregir los bugs.
- La calidad del software experimentada por el usuario final debe ser similar a la calidad experimentada por los ingenieros durante la fase de pruebas.
- La calidad del software no se reduce simplemente al proceso de eliminar las fallas encontradas en la etapa de pruebas.
- En base al estándar ISO/IEC 9126-1 la calidad del software abarca:
  - Funcionalidad.
  - Confiabilidad.
  - Usabilidad.
  - Eficiencia.
  - Mantenibilidad.
  - Portabilidad.

La fase de pruebas debe considerar todos estos factores también conocidos como características y atributos de calidad.

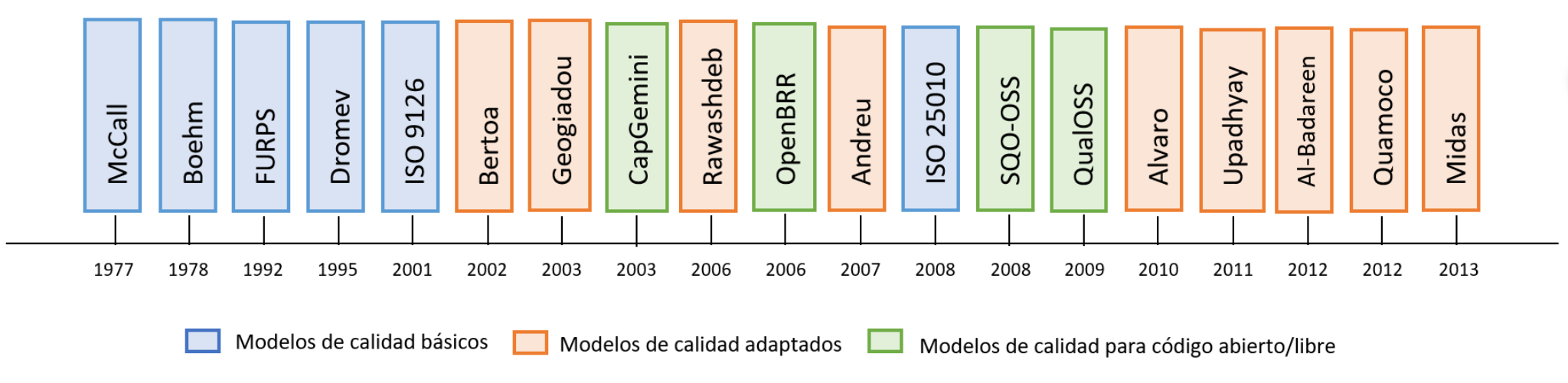
- En el 2011 el estándar ISO/IEC 9126 fue remplazado por el estándar 25010. La familia de normas ISO/IEC 25000 se encuentra compuesta por cinco divisiones.

Linea del tiempo del desarrollo de modelos de calidad



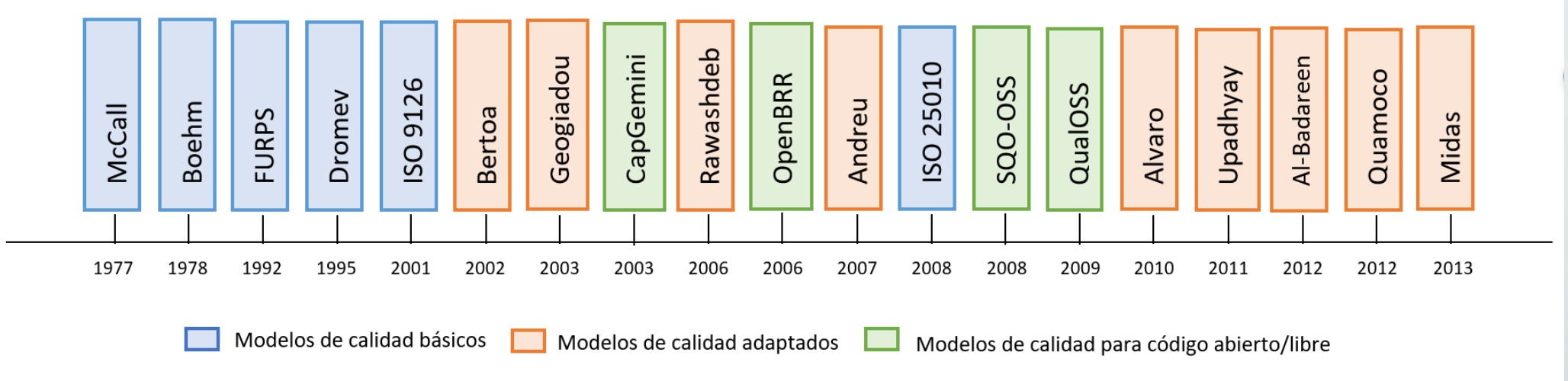
Modelos de calidad básicos. La mayoría de estos modelos fueron desarrollados entre los años de 1977 al 2001 y se caracterizan por ser jerárquicos en su estructura y estar diseñados para ser ajustados e implementados para cualquier producto de software.

Linea del tiempo del desarrollo de modelos de calidad



Modelos de calidad adaptados. Estos modelos surgieron a partir del año 2001 como consecuencia de las necesidades de la industria del software. Los modelos de calidad adaptados están basados en el modelo de calidad básico ISO 9126. Las principales características de estos modelos es que son específicos de un dominio en particular, se basan en la evaluación de componentes de software y la importancia de las características de calidad pueden variar en relación a un modelo general. Estos modelos son comúnmente utilizados para el software comercial basado en componentes.

Linea del tiempo del desarrollo de modelos de calidad



Modelos de calidad para software abierto. Dada las características del software abierto, el cual comúnmente se desarrolla en línea por una comunidad de desarrolladores interesados en mantener este tipo de software. A partir del año 2003 aparecieron algunos modelos de calidad orientados al software abierto.

Tarea 2

- **Equipo 1:**
  - Investigar los estándares asociados a las divisiones 2501n y 2503n del modelo de calidad ISO/IEC 2500n.
  - Realizar un resumen de los puntos más relevantes de las divisiones solicitadas.
  - Describir los aspectos y subaspectos de calidad definidos en el modelo 25010.

- **Equipo 2:**
  - Investigar los estándares asociados a las divisiones 2502n y 2504n del modelo de calidad ISO/IEC 2500n.
  - Realizar un resumen de los puntos más relevantes de las divisiones solicitadas.
  - Describir qué son las métricas de calidad y presentar ejemplos de su uso.

- Preparar un documento desarrollando los temas solicitados.
- Realizar una exposición a partir de la investigación realizada.
- Fecha de entrega 01 de septiembre de 2024.
- Fecha de presentación 02 de septiembre de 2024.
- Subir la tarea en forma y tiempo a Moodle.

Esfuerzo de las pruebas

- **Las pruebas no son utilizadas para mostrar la ausencia de bugs.** Para lograr este objetivo, una prueba debe ejecutar un programa en una variedad de posibles situaciones con cada valor posible de entrada y en todas las posibles condiciones.

- En la práctica no es viable el desarrollo de una prueba completa o exhaustiva. Debido a la gran variedad de posibles escenarios que se deben considerar. En la mayoría de los casos el número de pruebas requeridas serían infinitas.

Esfuerzo de las pruebas
Ejemplo – Primos rellenos

¿Cuál sería el total de posibles entradas para lograr una prueba exhaustiva?

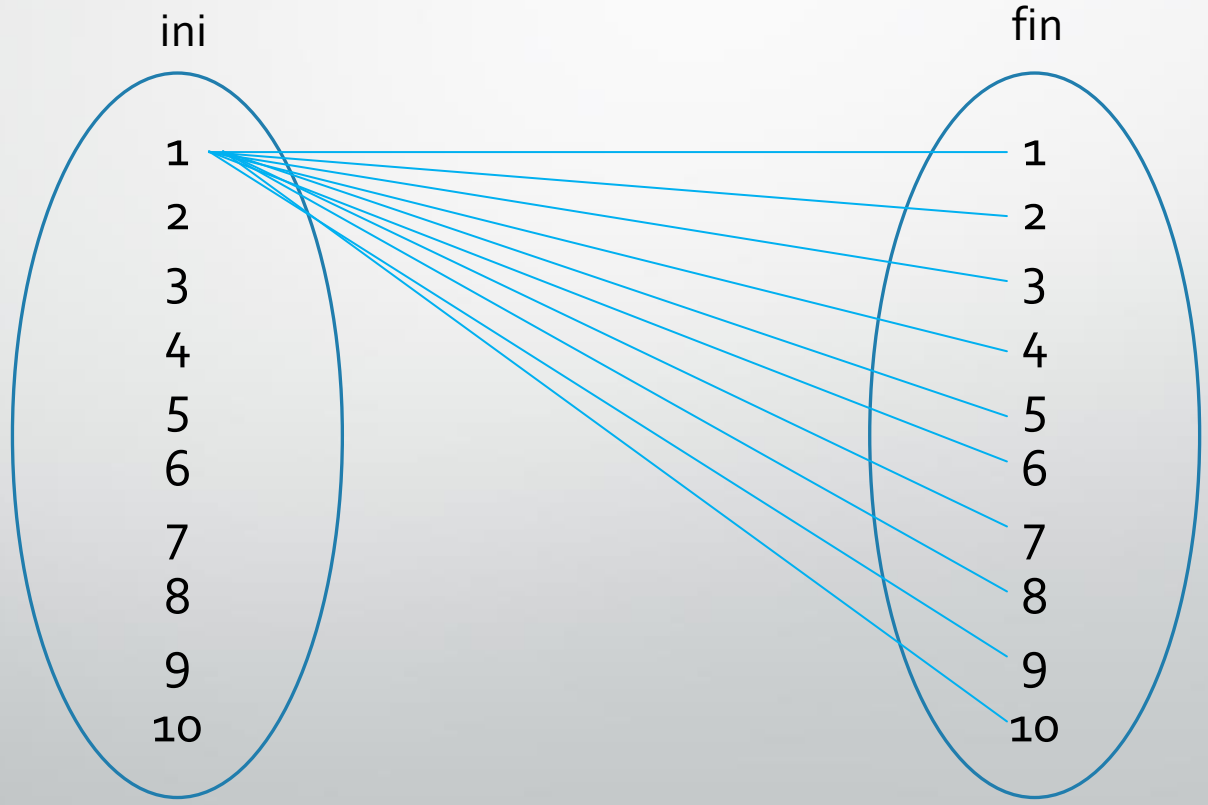
Descripción

- Un primo está definido como un número entero positivo que únicamente es divisible por la unidad y por sí mismo (para este problema consideraremos a la unidad también como primo). Existen primos especiales, como es el caso de los primos rellenos.
- Un número primo **n** formado de **m** dígitos ($n_1n_2...n_m$) se dice relleno si los números formados de los siguientes dígitos: n_1, n_1n_2, $n_1n_2n_3$, …, $n_1n_2...n_m$ son todos primos.
- El problema consiste en decir cuántos primos rellenos distintos entre **ini** y **fin** (incluyéndolos), donde 1 ≤ **ini** ≤ **fin** ≤ 100,000,000.
- Ejemplo:

Entrada	Salida
1, 1000	42
1, 2000	54
1, 10000	70

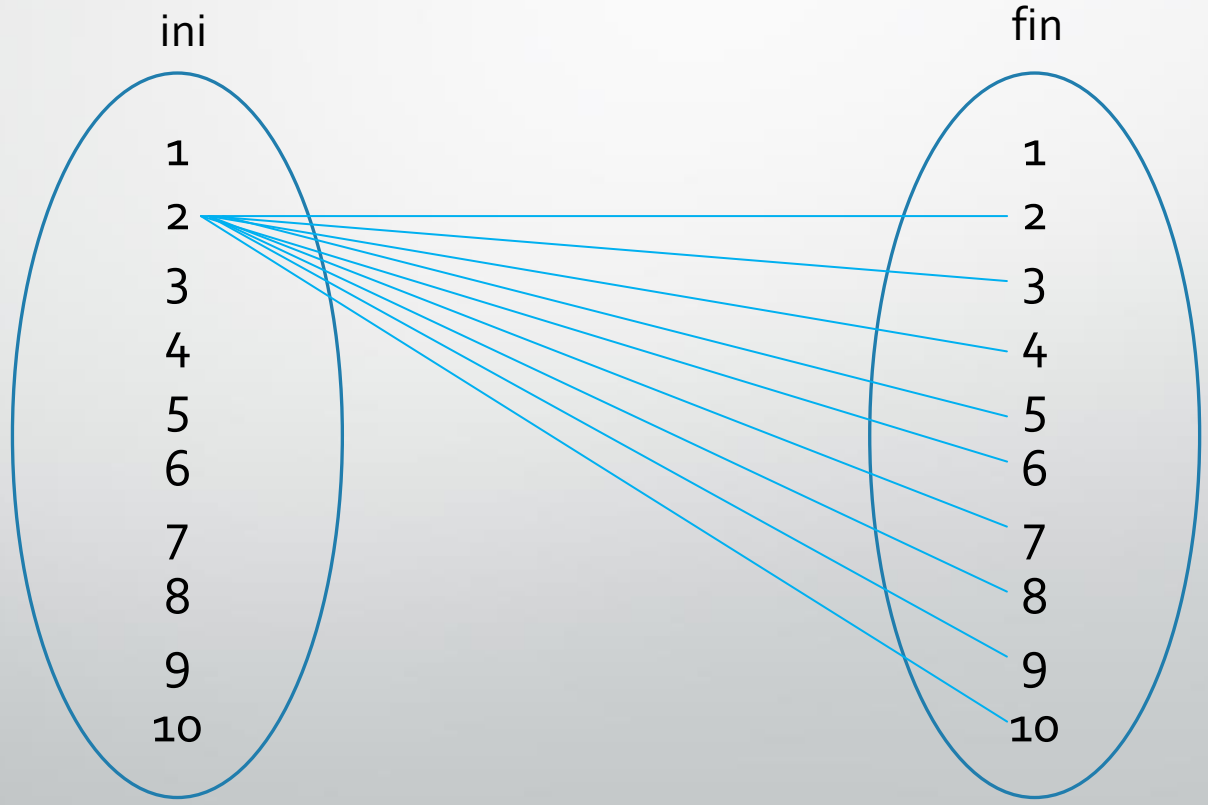
Esfuerzo de las pruebas
Ejemplo – Primos rellenos

Ejemplo con pruebas positivas, donde 1 ≤ **ini** ≤ **fin** ≤ 10~~0,000,000~~



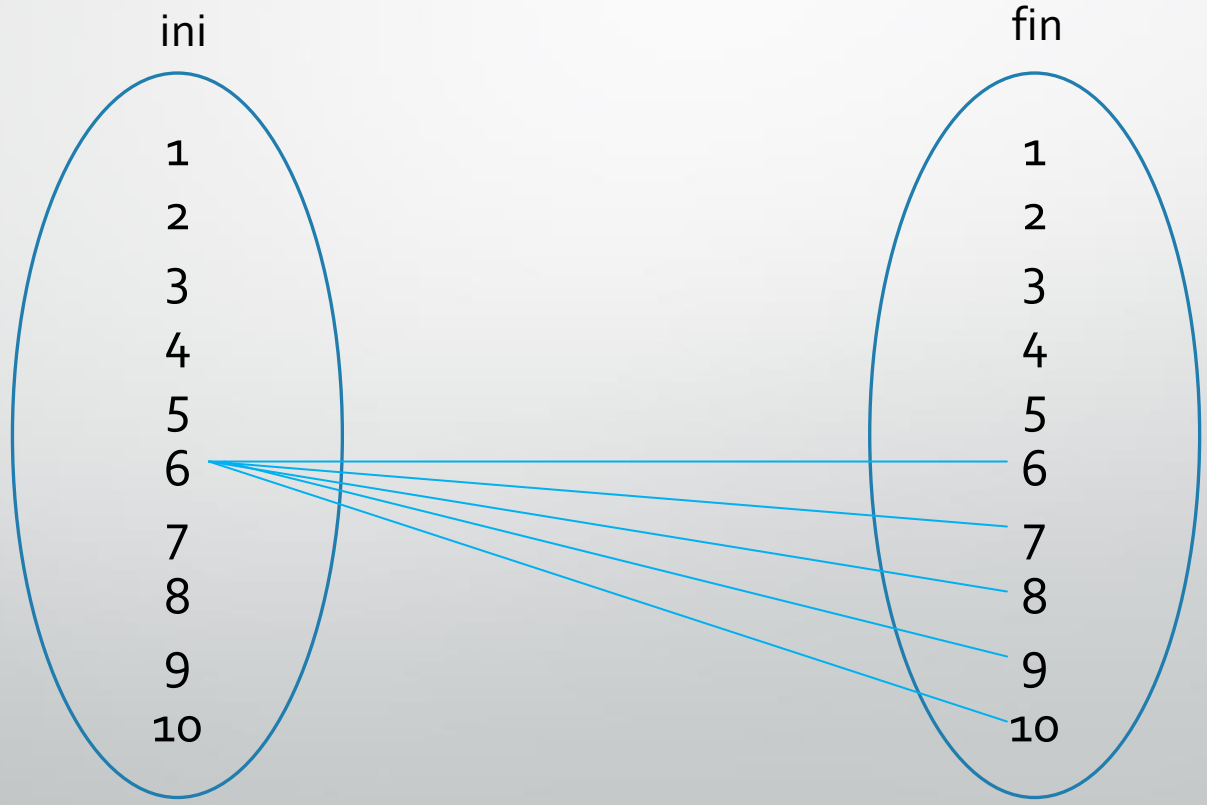
Esfuerzo de las pruebas
Ejemplo – Primos rellenos

Ejemplo con pruebas positivas, donde 1 ≤ **ini** ≤ **fin** ≤ 10~~0,000,000~~



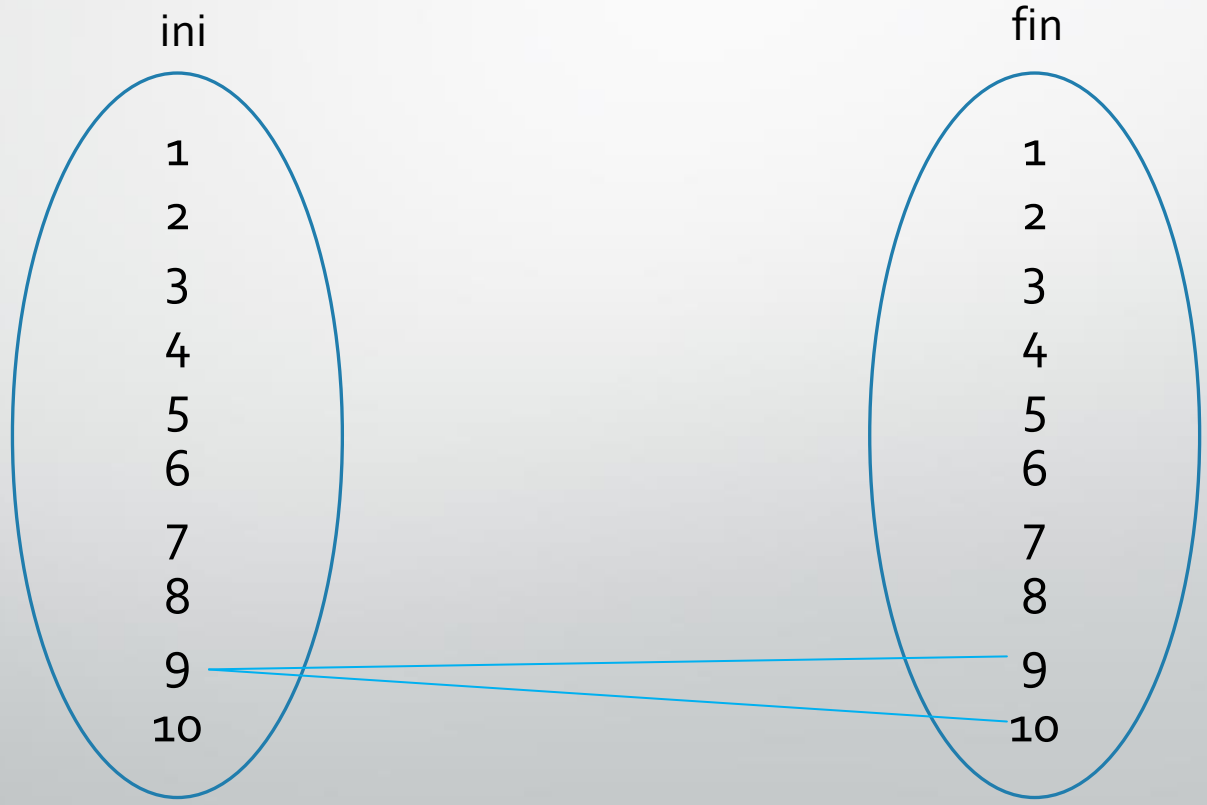
Esfuerzo de las pruebas
Ejemplo – Primos rellenos

Ejemplo con pruebas positivas, donde 1 ≤ **ini** ≤ **fin** ≤ ~~100,000,000~~



Esfuerzo de las pruebas
Ejemplo – Primos rellenos

Ejemplo con pruebas positivas, donde 1 ≤ **ini** ≤ **fin** ≤ 10~~0,000,000~~



Esfuerzo de las pruebas
Ejemplo – Primos rellenos

¿Cuál sería el total de posibles entradas para lograr una prueba exhaustiva?

Descripción

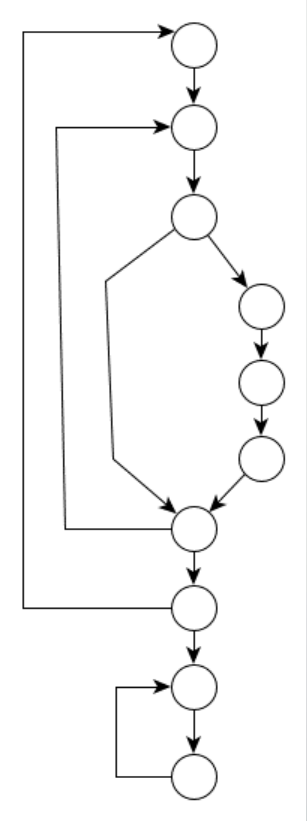
- Un primo está definido como un número entero positivo que únicamente es divisible por la unidad y por sí mismo (para este problema consideraremos a la unidad también como primo). Existen primos especiales, como es el caso de los primos rellenos.
- Un número primo **n** formado de **m** dígitos ($n_1n_2...n_m$) se dice relleno si los números formados de los siguientes dígitos: n_1, n_1n_2, $n_1n_2n_3$, …, $n_1n_2...n_m$ son todos primos.
- El problema consiste en decir cuántos primos rellenos distintos entre entre **ini** y **fin** (incluyéndolos), donde 1 ≤ **ini** ≤ **fin** ≤ 100,000,000.
- Ejemplo:

Entrada	Salida
1, 1000	42
1, 2000	54
1, 10000	70

$$Total\ de\ casos = \sum_{ini=0}^{fin} fin - ini$$

Esfuerzo de las pruebas

```
static void funcionX(int arreglo[])
{
  x = sizeof(arreglo) / sizeof(arreglo[0])
  for(int i = 0; i < x - 1; i++) {
    for(int j = i + 1; j < x - 1; j++) {
      if (arreglo[i] > arreglo[j]) {
        int tmp = arreglo[j];
        arreglo[j] = arreglo[i];
        arreglo[i] = tmp;
      }
    }
  }
  for(int i = 0;i < x; i++)
    System.out.print(arreglo[i]+"\n");
}
```

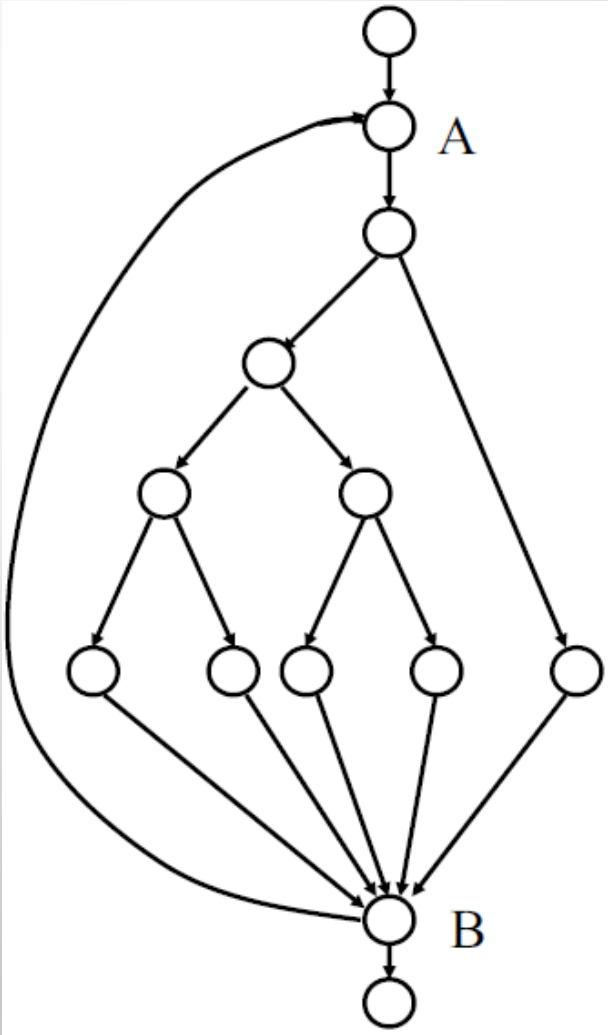


¿Cuántos datos y cuáles debería utilizar para probar de manera apropiada el código?

Ejemplo

- Considera un pequeño programa con un control de flujo sencillo que debe ser probado. El programa consiste de 4 estructuras condicionales que están parcialmente anidadas.
- Entre el punto A y B existe un ciclo que permite regresar del punto B al A.
- Sería extremadamente exhaustivo, si se desea realizar una prueba completa para los diferentes posibles flujos de control.
- Supongamos que 20 es el número máximo de vueltas para el ciclo repetitivo y que todos los caminos son independientes.

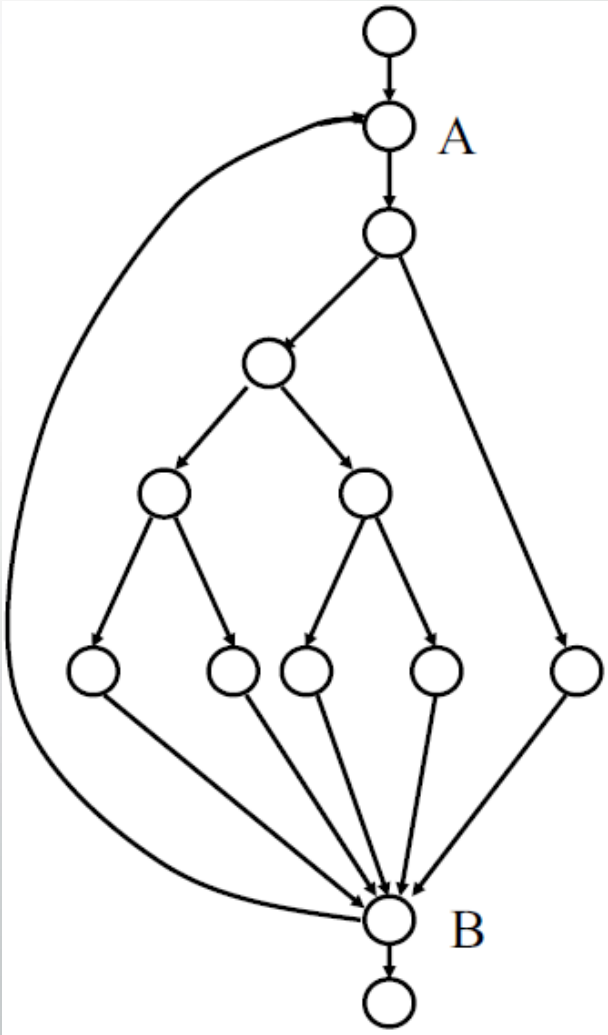
$$5^{20} + 5^{19} + 5^{18} + \ldots + 5^{1}$$



Ejemplo

- 5^1 casos de prueba serían el resultado de ejecutar cada posible camino simple, sin dar una segunda vuelta en la estructura repetitiva.
- El total de los casos de prueba para una sola vuelta en la estructura repetitiva sería $5 \times 5 = 5^2$.
- El total de casos de pruebas sería aproximadamente 100 trillones de sucesiones diferentes del programa.
- Considera que la prueba es realizada manualmente y que cada prueba requiere de 5 minutos para especificarla, ejecutarla y analizarla.

El tiempo de esta prueba sería de mil millones de años para concluirla.



Esfuerzo de las pruebas

- En la practica no es posible probar los programas de forma exhaustiva, incluso cuando se trata de pequeños programas. Solo es posible imaginar una parte de todos los posibles casos de prueba.

- Sin embargo, las pruebas representan una gran parte del esfuerzo de desarrollo.

- Es imposible definir de manera general el porcentaje de esfuerzo requerido para el desarrollo de las pruebas porque éste está en función de las características del proyecto.

Esfuerzo de las pruebas

Ejemplo del esfuerzo requerido en algunos proyectos desarrollados por una empresa alemana:

- Para algunos proyectos importantes que requieren del esfuerzo de más de 10 personas por año. La codificación y pruebas en conjunto requieren del 40% del esfuerzo y un 8% adicional para la integración.
- Proyectos que requieren de un mayor número de pruebas, por ejemplo, un sistema de seguridad crítica, el esfuerzo para las pruebas puede incrementar hasta en un 80% del esfuerzo total.
- En otro proyecto el esfuerzo de probar (testing) fue de 1.2 veces mayor al esfuerzo de codificar.

Esfuerzo de las pruebas

In Vincent Maraia's The Build Master (recommended) there's a helpful little chart on the size in lines of code for Windows NT:

Ship Date	Product	Dev Team Size	Test Team Size	Lines of code (LoC)
Jul-93	NT 1.0 (released as 3.1)	200	140	4-5 million
Sep-94	NT 2.0 (released as 3.5)	300	230	7-8 million
May-95	NT 3.0 (released as 3.51)	450	325	9-10 million
Jul-96	NT 4.0 (released as 4.0)	800	700	11-12 million
Dec-99	NT 5.0 (Windows 2000)	1,400	1,700	29+ million
Oct-01	NT 5.1 (Windows XP)	1,800	2,200	40 million
Apr-03	NT 5.2 (Windows Server 2003)	2,000	2,400	50 million

Book: The Build Master: Microsoft's Software Configuration Management Best Practices

Esfuerzo de las pruebas

- Comúnmente el esfuerzo es presentado en proporción entre el número de testers y el número de desarrolladores. Sin embargo, la proporción puede variar de un tester por cada 10 programadores (desarrolladores) e incrementar hasta 3 testers por cada programador.

- ***La conclusión es que el esfuerzo requerido para las pruebas puede variar enormemente de un proyecto a otro.***

Primer avance de proyecto

Revisar las instrucciones en Moodle para realizar esta primer entrega de avances.

Fecha de la presentación: 02 de septiembre de 2024.

¿Cómo escribir un buen SRS?

Un buen SRS debe cumplir con varias características clave:

- **Correcto:** Es importante asegurarse de que el SRS siempre refleje la funcionalidad y especificación del producto.
- **Sin ambigüedades:** Es mejor ser demasiado específico que ambiguo. El SRS no es una obra maestra literaria, por lo que incluso las reglas estilísticas más básicas pueden ser ignoradas en nombre de la claridad.
- **Completo:** Nunca es una buena idea omitir cualquier característica solicitada por el cliente.
- **Consistente:** Todos los acrónimos y definiciones deben ser utilizados de manera consistente en todo el SRS.
- **Clasificación por importancia y/o estabilidad:** El tiempo es a menudo un recurso escaso durante el proceso de desarrollo, por lo que es una buena idea clasificar los requisitos según su importancia y estabilidad.
- **Verificable:** Debe haber un método de verificación para cada requisito.
- **Modificable:** Los cambios en los requisitos deben hacerse de manera sistemática, y debe tenerse en cuenta su impacto en otros requisitos.
- **Rastreable:** Todos los requisitos deben ser trazables desde su origen.