Un Marco Metodológico para Evaluar Técnicas y Herramientas para Pruebas del Software

Tanja E. J . Vos
Centro de Métodos de Producción de Software
Universidad Politecnica de Valencia
Valencia, Spain
tvos@pros.upv.es

Beatriz Marín
Universidad Diego Portales,
Santiago, Chile
beatriz.marin@mail.udp.cl

Maria Jose Escalona
Ingeniería Web y Testing Temprano (IWT2),
Universidad de Sevilla,
Sevilla, Spain
mjescalona@us.es

Resumen

Actualmente existe una necesidad real en el sector industrial de disponer de conocimiento para decidir qué técnicas de pruebas deben usarse según los objetivos de pruebas, y para conocer cómo de usables (efectivas, eficientes y satisfactorias) pueden llegar a ser estas técnicas. Sin embargo, estas guías en realidad no existen. Podríamos plantearnos como medio para conseguirlas el realizar estudios comparativos de evaluación de técnicas de pruebas y herramientas basadas en casos de estudio, sin embargo, estos estudios también son poco viables por falta de disponibilidad de dichos casos.
En este trabajo, daremos un primer paso a crear una primera aproximación a un marco de trabajo de evaluación que permita simplificar el diseño de los casos de estudio a comparar en las herramientas de pruebas de software, haciendo los resultados lo más precisos, legibles y fáciles de comparar.

Abstract

There exists a real need in industry to have guidelines on what testing techniques use for different testing objectives, and how usable (effective, efficient, satisfactory) these techniques are. Up to date, these guidelines do not exist. Such guidelines could be obtained by doing secondary studies on a body of evidence consisting of case studies evaluating and comparing testing techniques and tools. However, such a body of evidence is also lacking. In this paper, we will make a first step towards creating such body of evidence by defining a general methodological evaluation framework that can simplify the design of case studies for comparing software testing tools, and make the results more precise, reliable, and easy to compare.

1. Introducción

Para realizar las pruebas de software se necesita tomar decisiones informadas acerca de qué técnicas usar en situaciones específicas, y estimar el tiempo y esfuerzo necesarios para aplicarlas. Sin embargo, actualmente no es posible encontrar guías claras para estas tareas debido principalmente a la falta de estudios empíricos que sirvan como experiencias documentadas para estudios secundarios de ingeniería de software basada en evidencia (en inglés Evidence-Based Software Engineering – EBSE) [KDJ04].

En este contexto, es necesario contar con más estudios que evalúen y comparen empíricamente técnicas y herramientas para realizar pruebas de software [JMV04, RAT+06]. Sin embargo, para asegurar que la evidencia de estos estudios pueda resultar en guías adecuadas, los casos de estudios evaluados deberían:

- Involucrar sistemas y sujetos realistas, y no programas de ejemplo y estudiantes como es el caso de la mayoría de estudios realizados.
- Ser realizados con rigurosidad para asegurar que cualquier beneficio identificado durante el estudio es claramente derivado de la técnica de pruebas estudiada.
- Asegurar que diferentes estudios pueden ser comparados.

Aunque este tipo de investigación es caro, difícil, y consume demasiado tiempo, es fundamental [FPG94].

Desafortunadamente, las empresas a menudo se muestran reacias a participar en casos de estudio. Algunas no están dispuestas a probar nuevos enfoques que quizás no han sido probados. Otras se preocupan de que se podrían revelar fallas críticas, métricas pobres o bajo rendimiento. Otras empresas no están dispuestas a permitir a los investigadores realizar los estudios por temor de perder información confidencial, que ellos hagan más lento el trabajo del equipo, o simplemente no sepan cómo hacerlo. Independientemente de los motivos, es necesario superar estas barreras con el fin de avanzar y crear el cuerpo de evidencia necesario.

En este trabajo, se propone un marco *metodológico* general para reducir algunas de las barreras de entrada para realizar casos de estudio. Este marco metodológico simplificará el diseño de los casos de estudio para comparar técnicas de pruebas de software, asegurando que la mayoría de las guías definidas para realizar estudios empíricos han sido seguidas. Además, si los casos de estudio son ejecutados con un diseño similar (es decir, por la instanciación del marco propuesto), será posible replicar los estudios, y con la evidencia generada, será más fácil comparar y efectivamente agregar estos estudios en estudios secundarios.

El marco que se presenta en este trabajo ha evolucionado durante los últimos años mediante la realización de casos de estudio realizados para evaluar técnicas de pruebas [VME+12]. La necesidad de contar con un marco como el que se describe en este trabajo ha surgido hace algunos años durante la ejecución del proyecto EvoTest (IST-33472, 2007-2009, vea [Vos09]) y continuado durante la ejecución del proyecto FITTEST (ICT-257574, 2010-2013, vea [Vos10]). Ambos proyectos tienen como objetivo el desarrollo de herramientas de pruebas que durante el proyecto deben ser evaluadas en entornos industriales. En estos proyectos, se necesita un marco que pueda ser instanciado para cualquier tratamiento, sujeto y objeto y que simplifique el diseño de los estudios evaluativos sugiriendo preguntas y medidas relevantes. Dado que este marco no existe, en este trabajo se presenta un marco asegurándonos de seguir las guías y directrices encontradas en la literatura. Hasta la fecha, el marco se ha utilizado exitosamente en varios casos de estudio de los proyectos EvoTest y FITTEST.

2. Trabajo Relacionado

La mayoría del trabajo relacionado existente presenta marcos *organizacionales* y guías, como por ejemplo listas de pasos que deben ser llevados a cabo, o advertencias de que los estudios deben ser diseñados cuidadosamente y que los factores que lleven a confusión deben ser minimizados. Con la excepción de [EH+06], que se limita a la inyección de fallos, no se ha encontrado ningún trabajo que especifique cómo evaluar técnicas de pruebas de software, cómo deben ser definidas las preguntas de investigación, qué variables deben ser medidas, cuáles pueden ser las amenazas a la validez de los estudios, etc.

Lott y Rombach [LR96] describen un esquema de caracterización para experimentos de pruebas de software. Este esquema es similar al esquema general presentado en [BSH86], pero adaptado para evaluar técnicas de prueba. Este esquema ayuda a los usuarios a organizar y evaluar el diseño de un experimento. Sin embargo, no ofrece ninguna ayuda concreta sobre la forma en que los distintos componentes de un experimento pueden ser diseñados y qué puede ser medido.

Do et al. [DER05] definen SIR, un Repositorio de Artefactos de Infraestructura de Software (sir.unl.edu), para apoyar experimentos controlados con técnicas pruebas de software. La principal contribución de su trabajo es un conjunto de programas que pueden ser utilizados para evaluar las pruebas técnicas. No se presentan directrices metodológicas claras.

El trabajo de Eldh et. al [EH+06] describe un marco para la comparación de técnicas de prueba basadas en inyección de fallos. Los pasos del marco son: preparar muestras de código con fallos conocidos mediante inyección de fallos; seleccionar una técnica de pruebas; realizar el experimento y recopilar datos; analizar los datos y repetir el experimento si es necesario. Este trabajo describe los desafíos industriales para cada paso. Este trabajo describe un marco metodológico interesante, pero se limita a inyección de fallos.

El marco organizacional DESMET [KLL97], al igual que el marco presentado en este trabajo, ha sido especialmente desarrollado para evaluar métodos y herramientas dentro de las empresas, no está especialmente dirigido a investigadores sino a los vendedores de herramientas, ingenieros de software que deseen evaluar una propuesta de modificación, etc.

3. Un marco para evaluar herramientas de pruebas

Imagine que la empresa C quiere evaluar una técnica o herramienta de pruebas T para ver si es usable y vale la pena incorporar esta técnica o herramienta en sus procesos de pruebas. Las siguientes secciones ayudan a definir un caso de estudio.

3.1 Objetivo del estudio - ¿Qué alcanzar?

El marco general se centra en efectividad, eficiencia y satisfacción subjetiva. De esta manera, las preguntas de investigación para cada caso de estudio corresponden a instancias de:

RQ_1 ¿Cómo contribuye T a la efectividad de las pruebas (capacidades de encontrar fallos) cuando es usada en entornos de pruebas reales de C y comparada con las actuales prácticas de pruebas usadas en C?

RQ_2 ¿Cómo contribuye T a la eficiencia de las pruebas cuando es usada en entornos de pruebas reales de C y comparada con las actuales prácticas de pruebas usadas en C?

RQ$_3$ ¿Qué tan satisfechos están los profesionales de pruebas de C durante el aprendizaje, instalación, configuración, y uso de T cuando es usado en entornos de pruebas reales?

3.2 Sujetos - ¿Quién aplica las técnicas/herramientas?

Idealmente, los sujetos son trabajadores de C. Los sujetos deberían ser las personas que normalmente utilizan las técnicas o herramientas que están siendo comparadas con T. Si por alguna razón esto no es posible (por ejemplo, debido a la falta de tiempo y recursos, la herramienta es un prototipo académico, etc.), entonces investigadores o desarrolladores de herramientas pueden evaluar las herramientas. Sin embargo, esto significa que la satisfacción subjetiva no puede ser medida y no se pueden obtener resultados de las capacidades de la herramienta dentro de un entorno industrial.

3.3 Objetos - ¿Cuáles son los proyectos piloto?

El sistema sometido a pruebas debería ser un sistema que es típico de C (es decir, manteniendo la manera en que el software es desarrollado, la forma en que se prueba, idiomas utilizados, etc.). Además, la información disponible de este sistema debería ser determinada o medida para hacer la comparación con T. Para ello, las siguientes preguntas deben ser respondidas:

S1. ¿Se tendrá acceso a un sistema con fallos conocidos? ¿Qué información se tiene acerca de esos fallos?

S2. ¿Se puede inyectar fallos al sistema?

S3. ¿C recopila datos de sus proyectos como práctica estándar? ¿Qué datos son éstos? ¿Están estos datos disponibles para la comparación? ¿Se tiene una línea base de la empresa? ¿Se tiene acceso a proyectos similares?

S4. ¿La empresa C tiene suficiente tiempo y recursos para realizar varias rondas de pruebas? O de manera más concreta:

- ¿Está la empresa C dispuesta a hacer una nueva testsuite TS_N con alguna técnica/herramienta T_{known} ya utilizada en la empresa o T_{unkown} que es nueva en la empresa?
- ¿Se puede utilizar para comparar una testsuite TS_C existente? ¿Se conocen las técnicas que fueron utilizadas para crear esa testsuite? Y ¿cuánto tiempo tomó su desarrollo?

3.4 Variables - ¿Qué datos recolectar?

Las variables independientes consideradas son: método de pruebas T usado; Complejidad de los sistemas industriales, nivel de experiencia de los ingenieros de C que van a hacer las pruebas.

La variable dependiente está orientada a medir eficacia, eficiencia y satisfacción. La siguiente es una lista de las mediciones que se pueden realizar. Para una instanciación específica de este marco en una empresa, algunas de estas variables podrían no ser aplicables.

1. ***Midiendo Efectividad***
    a) Número de casos de prueba diseñados o generados.
    b) Número de casos de prueba inválidos generados.
    c) Número de casos de prueba generados repetidos.
    d) Número de deficiencias observadas.
    e) Número de fallos encontrados.
    f) Número de falsos positivos.
    g) Número de falsos negativos.
    h) Tipo y causa de los fallos que fueron encontrados.
    i) Cobertura alcanzada (estimada o medida).

2. ***Midiendo Eficiencia***
    j) Tiempo necesitado para aprender el método de pruebas T.
    k) Tiempo necesitado para diseñar o generar los casos de pruebas.
    l) Tiempo necesitado para establecer la infraestructura de pruebas específica para T (instalar, configurar, desarrollar controladores de pruebas, etc.).
    m) Tiempo necesitado para probar y observar deficiencias (es decir, planificación, implementación, y ejecución).
    n) Tiempo necesitado para identificar tipos de fallos y causas para cada deficiencia observada.

3. ***Midiendo Satisfacción Subjetiva***

    o) Cuestionario de Puntuación de Usabilidad del Sistema (System Usability Score – SUS por sus siglas en inglés) [Bro96] consistente de 10 preguntas con escala Likert de 5 puntos y un puntaje total.
    p) 5 reacciones (a través de cartas de reacción) que serán usadas para crear una nube de palabras y diagramas Ven.
    q) Tarjetas de reacciones faciales emocionales durante las entrevistas (caras serán evaluadas en una escala Likert de 5 puntos desde totalmente en desacuerdo a totalmente de acuerdo).
    r) Opiniones subjetivas acerca de T.

Se ha decidido utilizar el cuestionario SUS porque este sencillo cuestionario entrega los resultados más fiables [BKM08]. Se ha complementado el cuestionario SUS con otras medidas debido a que los cuestionarios tienen limitaciones conocidas. En una revisión, Hornbaek [Horn06] concluye que las medidas de satisfacción deben extenderse más allá de los cuestionarios. Se han ampliado los cuestionarios con nuevos métodos para medir la satisfacción desarrollados por Microsoft, como por ejemplo las tarjetas reacción y cuestionarios de caras [BM02].

3.5 Protocolo - ¿Cómo ejecutar el estudio?

En la Figura 1 se presentan los pasos necesarios. Si se puede inyectar fallos en el sistema, hay que tener cuidado que [MX05]:

- Los fallos sembrados son similares a los fallos que se producen de forma natural en los programas reales debido a los errores cometidos por los desarrolladores.
- Deberían inyectarse bastantes fallos, es decir, un número suficiente de casos de cada tipo de fallo se ha sembrado.

El tipo y el procedimiento para ejecutar cada caso de estudio depende de las respuestas dadas a las preguntas de la Sección 3.4 . Esto se traduce en 7 posibles escenarios de casos de estudio (ver Sección 3.4 (S4) para recordar el significado de la T_{known}, $T_{unknown}$, TS_C, TS_N):

Escenario 1. Consiste en una evaluación cualitativa. Dado que no se sabe cuántos errores hay, no se puede comparar con otras técnicas, ni tampoco se puede realizar una línea base de la empresa, no se puede hacer una evaluación cuantitativa. Sin embargo, estudiar y reportar las medidas encontradas de eficacia, eficiencia y satisfacción subjetiva se llevará a cabo durante entrevistas semi-estructuradas con profesionales de pruebas.

Escenario 2. Consiste en Escenario 1 y análisis cuantitativo basado en la línea base de la empresa. La medida en que esto es posible y cuán válidas son las conclusiones dependerá de los datos que se presentan en la línea base de la empresa.

Escenario 3. Consiste en (Escenario 1 o Escenario 2) y análisis cuantitativo de la Tasa de Detección de Fallos (FDR) respecto al conjunto conocido de fallos.

Escenario 4. Consiste en (Escenario 1 o Escenario 2) y comparación cuantitativa de T y TS_C. Dado que TS_C ya existe, hay algunas medidas que no se pueden comparar (por ejemplo, en relación a la creación/diseño de la suite de pruebas, etc.), que serán cubiertas con el análisis del Escenario 1.

Escenario 5. Consiste en Escenario 4 y FDR de T y TS_C. Este escenario agrega una comparación cuantitativa de la tasa de detección de fallos de T con TS_C al Escenario 4.

Escenario 6. Consiste en (Escenario 1 o Escenario 2) y comparación cuantitativa de T y (T_{known} or $T_{unknown}$).

Escenario 7. Consiste de Escenario 6 y FDR de T y (T_{known} or $T_{unknown}$).

4. Casos de aplicación del framework

Esta sección describe tres instancias del framework. Las experiencias son diferentes pero aun así, la instanciación del framework resulta sencilla. En los sub-apartados siguientes se muestra cómo hemos aplicado el framework de manera efectiva en todos estos ejemplos.

4.1. Pruebas estructurales basadas en búsquedas

En el trabajo [VBF+10] se presenta una instancia del framework sobre un caso de estudio. El objetivo de este trabajo era investigar la escalabilidad de las pruebas estructurales basadas en búsquedas desarrolladas en el proyecto EvoTest [Vos09] y que se automatizaron con una herramienta llamada ETF (Evolutionary Testing Framework).

La herramientas fue evaluada con dos compañías (Daimler[1] y Berner&Mattner[2]) quienes participaron en el estudio por ser socios del proyecto EvoTest.

Las cuestiones planteadas en la investigación relacionadas con la efectividad, eficiencia y satisfacción del usuario fueron:

---

[1] http://www.daimler.com

[2] http://www.berner-mattner.com

- En comparación con las pruebas aleatorias, EFT resulta más efectivo y eficiente encontrando casos de pruebas en sistemas reales.
- La cantidad de tiempo, esfuerzo y conocimiento necesario para configurar y usar ETF lo hace complejo en los entornos industriales.

Este caso de estudio se dirigió hacia los técnicos de pruebas empleados en las empresas participantes (sujetos). En el caso del objeto del estudio, fueron funciones C seleccionadas de sistemas reales.

Sin embargo, dado que las compañías no podían ofrecernos información sobre los bugs existentes ni podían incluir fallos, la respuesta a las preguntas S1 y S3 de la sección 3.4 fue NO. La respuesta a la cuestión S4 fue sí pero sólo con otra técnica que pude ser utilizada con un mínimo esfuerzo humano, como es por ejemplo la técnica de pruebas aleatorias.

Las variables de la sección 3.5 fueron medidas a través del número de casos de pruebas, el grado de código cubierto, el tiempo necesario para personalizar ETF, tiempo para generar los casos de prueba, tiempo para probar el sistema y la satisfacción subjetiva y cualitativa obtenida en entrevistas informales con los equipos en las empresas.

Dado que no se podían prever los fallos a encontrar puesto que el software estaba bajo producción, se midieron los valores de progreso para conseguir una medida cualitativa de la calidad de los casos de prueba.

La instancia de ***do*** de la sección 3.6 y Figura 1 consiste en: (1) Instalar y configurar ETF de acuerdo a su manual. Durante estas actividades, el trabajo diario medía y contenía una lista de tareas (que incluye aspectos como fecha, tiempo y descripción) que eran ejecutadas de acuerdo a ETF (por ejemplo, instalación, configuración, encontrar un conjunto apropiado de la evolución, etc.) (2) Ejecutar cada búsqueda 30 veces para asegurar las medidas conseguidas y recopilar los datos a las variables seleccionadas. (3) Tener entrevistas informales sobre la adecuación y aceptabilidad en el entorno empresarial.

4.2. Pruebas funcionales basadas en búsquedas

En [VFW+12] se usó una instancia del framework para evaluar un caso de estudio cuyo principal objetivo era ver la aplicabilidad de pruebas funcionales basadas en las búsquedas.

Dado que las pruebas funcionales no están completamente automatizadas (como en el caso de las pruebas estructurales de la sección anterior) y que, de nuevo las empresas indicaron que al ser un sistema en producción sería complejo encontrar fallos y que no se podrían generar más fallos. De esta forma, las preguntas planteadas fueron:

Efectividad:

- ETF aplicado al mundo real, permite generar mejores casos de pruebas en comparación con otras técnicas como las pruebas aleatorias.

- ETF es más efectivo encontrando errores cuando se aplica en entornos reales para pruebas de caja negra que para pruebas aleatorias.

Satisfacción del usuario y eficiencia:

- Es posible utilizar ETF sin un conocimiento detallado del evolutivo para buscar las pruebas de interés.

- Después de la instalación de ETF, la cantidad de tiempo y esfuerzo que se necesita para configurarlo es abordable en un entorno industrial.

En este caso, el sujeto estaba compuesto por tres testers de sistemas empotrados en cada una de las dos empresas. Los testers ya tenían un conocimiento previo de los principios de pruebas evolutivas. Los objetos eran los sistemas de control integrados reales en el campo del automóvil.

Una vez más, las empresas no podrían compartir información acerca de los errores existentes, conjuntos de pruebas existentes, ni podían añadir fallos ni pruebas por encontrarse con sistemas en explotación, por lo que la respuesta a las preguntas de S1 a S4 de la sección 3.4 fueron NO. Así, el escenario de la sección 3.6 se corresponde en este caso con el número 1. La calidad de los casos de pruebas se analizaba en función de los resultados conseguidos. Por último, indicar que se añadió una comparación cuantitativa de las pruebas al azar para tener una línea de base para la comparación de los algoritmos de búsqueda subyacentes.

Las variables que se midieron fueron el número de casos de prueba, el número de casos de pruebas no válidos, el número de fallos encontrados, el tiempo necesario para configurar ETF, el tiempo para generar los casos de prueba, el tiempo para testear el sistema y la evaluación subjetiva conseguidas de las entrevistas en el entorno industrial.

Como no se esperaba encontrar fallos en el sistema, usamos la medida de la conveniencia de las pruebas en función de la calidad de las pruebas.

La instancia de ***do*** de la sección 3.6 y Figura 1 consistió en: (1) Instalar y configurar de acuerdo al manual de usuario de ETF. El mantenimiento de las trabajos diarios incluía las tareas (incluyendo fecha, tiempo y descripción) que eran ejecutadas para configurar el manual de ETF (por ejemplo, tareas para encontrar un conjunto apropiado de parámetros, etc.); (2) Implementar los componentes de los casos de estudio (por ejemplo, especificaciones individuales y los drivers de prueba). (3) Definir, redefinir e implementar las funciones de completitud y validar su adecuación para la gestión de los requisitos y los trabajos diarios (4) Ejecutar las búsquedas 30 veces para dar un valor estadístico adecuado y recopilar los datos. (5) Tener entrevistas con los testers en las empresas para conocer su opinión.

4.3. Web testing de aplicaciones AJAX

En [MRT08] el framework se instanció para evaluarlo en base a 4 técnicas de prueba que se ejecutan de manera frecuente en pruebas web: pruebas basadas en modelos, técnicas basadas en medidas de cobertura, técnicas de caja negra y pruebas basadas en estado.

Las técnicas fueron evaluadas por académicos y los objetos de evaluación fueron aplicaciones web obtenidas en un libro de estudio de aplicaciones industriales. Dado que el estudio fue realizado por académicos, no se pudieron hacer medidas subjetivas de satisfacción así que las preguntas de investigación fueron:

- ¿Que efectividad tienen la pruebas de técnicas de pruebas en Web para la detección de fallos?
- ¿Cuál es el esfuerzo requerido para aplicar cada una de las técnicas?

Teniendo en cuenta que no era software industrial, podíamos incluir fallos así que el escenario de las sección 3.6 en este caso se corresponde con el número 3. Las variables que se midieron fueron: número de fallos encontrados, cobertura de los casos de uso, tipos y criticidad de los fallos que se encontraron, tamaño de los paquetes de prueba, tiempo (en

horas/hombre) necesarias para testear la infraestructura específica así como la complejidad de los paquetes de prueba.

La instanciación de ***do*** de la sección 3.6 consistió en: (1) Inyectar fallos (esta actividad era realizada por una persona diferente al que probaba) dentro de las aplicaciones web originales, intentando simular los errores de la programación usando la taxonomía o defectos de [MRT08]. Estos cambios no producían errores graves en la aplicación pero fallaban en el comportamiento esperado de la misma; (2) Aplicar las técnicas de prueba Web seleccionada para encontrar estos fallos derivando paquetes de prueba para cada uno de ellos; (3) Aplicar cada paquete de pruebas en las aplicaciones.

5. Conclusiones

En este trabajo se ha presentado un framework metodológico para evaluar técnicas de pruebas de software. El objetivo esencial del framework es permitir al equipo de pruebas facilitar la definición de los casos de estudio instanciando dicho framework, con la seguridad de que dicho framework se basa en trabajos empíricos. Además, como todos los casos de estudio se ejecutan en base al mismo diseño, será más sencillo comparar todos los resultados obtenidos permitiendo a los investigadores disponer de un grupo de experiencias que permitan mejorar el campo de investigación de las pruebas de software.

En el trabajo se han presentado tres ejemplos de aplicación exitosa del framework que nos ha permitido evaluar la aplicabilidad y efectividad de la propuesta. Recientemente se ha utilizado el framework para 3 estudios más que están pendientes de publicación. Nuestros trabajos futuros van encaminados hasta esta dirección, intentando conseguir más ejemplos para la validación del framework, redefinirlo y comenzar a crear nuestro repositorio de experiencias que mejoren la investigación en las pruebas de software.

Agradecimientos

Este trabajo ha sido parcialmente financiado por los proyectos: FITTEST (ICT257574, UE, 2010-2013), CaSA-Calidad (TIN2010-12312-E, Ministerio de Ciencia e Innovación) y Fondecyt TESTMODE (11121395, 2012-2015).

Bibliografía

[BKM08] A. Bangor, P. Kortum, and J. Miller, “An empirical evaluation of the system usability scale,” International Journal of Human-computer Interaction, vol. 24, pp. 574–594, 2008.

[BM02] J. Benedek and T. Miner, “Measuring desirability: New methods for measuring desirability,” in Proc of the Usability Professionals Association Conf, 2002.

[Bro96] J. Brooke, “Sus: a ’quick and dirty’ usability scale,” in Usability Evaluation in Industry. Taylor and Francis, 1996.

[BSH86] V. Basili, R. Selby, and D. Hutchens, “Experimentation in software engineering,” IEEE TSE, vol. 12, pp. 733–743, 1986.

[DER05] H. Do, S. Elbaum, and G. Rothermel, “Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact,” ESE, vol. 10, no. 4, pp. 405–

435, 2005.

[EH+06] S. Eldh, H. Hansson, S. Punnekkat, A. Pettersson, and D. Sundmark, “A framework for comparing efficiency, effectiveness and applicability of software testing techniques,” TAICPart, pp. 159–170, 2006.

[FPG94] N. Fenton, S. Pfleeger, and R. Glass, “Science and substance: a challenge to software engineers,” Software, IEEE, vol. 11, no. 4, pp. 86 –95, Jul. 1994.

[Horn06] K. Hornbaek, “Current practice in measuring usability: Challenges to usability studies and research,” Int. J. Hum.-Comput. Stud., vol. 64, pp. 79–102, February 2006.

[JMV04] N. Juristo, A. Moreno, and S. Vegas, “Reviewing 25 years of testing technique experiments,” ESE., vol. 9, no. 1-2, pp. 7–44, 2004.

[KDJ04] B. Kitchenham, T. Dyba, and M. Jorgensen, “Evidence-based software engineering,” in Proc of ICSE. IEEE, 2004, pp. 273–281

[KLL97] B. Kitchenham, S. Linkman, and D. Law, “Desmet: a methodology for evaluating software engineering methods and tools,” Computing Control Engineering Journal, vol. 8, no. 3, pp. 120 –126, Jun. 1997.

[LR96] C. Lott and H. Rombach, “Repeatable software engineering experiments for comparing defect-detection techniques,” ESE, vol. 1, pp. 241–277, 1996.

[MRT08] A. Marchetto, F. Ricca, and P. Tonella, “A case study based comparison of web testing techniques applied to ajax web applications,” International Journal on Software Tools for Technology Transfer, vol. 10, pp. 477–492, 2008

[MX05] A. Memon and Q. Xie, “Studying the fault-detection effectiveness of gui test cases for rapidly evolving software,” IEEE TSE , vol. 31, no. 10, pp. 884 – 896, oct. 2005.

[RAT+06] P. Runeson, C. Andersson, T. Thelin, A. Andrews, and T. Berling, “What do we know about defect detection methods?” IEEE Softw., vol. 23, no. 3, pp. 82–90, 2006

[VME+12] T. Vos, B. Marín, MJ Escalona, A. Marchetto: A Methodological Framework for Evaluating Software Testing Techniques and Tools. QSIC 2012: 230-239, 2012.

[VBF+10] T. Vos, A. Baars, F. Lindlar, P. Kruse, A. Windisch, and J. Wegener, “Industrial scaled automated structural testing with the evolutionary testing tool,” in ICST, 2010, pp. 175–184.

[VFW+12] T. Vos, F. Lindlar, B. Wilmes, A. Windisch, A. Baars, P. Kruse, H. Gross, and J. Wegener, “Evolutionary functional black-box testing in an industrial setting,” Software Quality Journal, pp. 1–30, 2012.

[Vos09] T. Vos, “Evolutionary testing for complex systems,” ERCIM News, vol. 2009, no. 78, 2009.

[Vos10] T. Vos, “Continuous evolutionary automated testing for the future internet,” ERCIM, vol. 2010, no. 82, pp. 50–51, 2010.

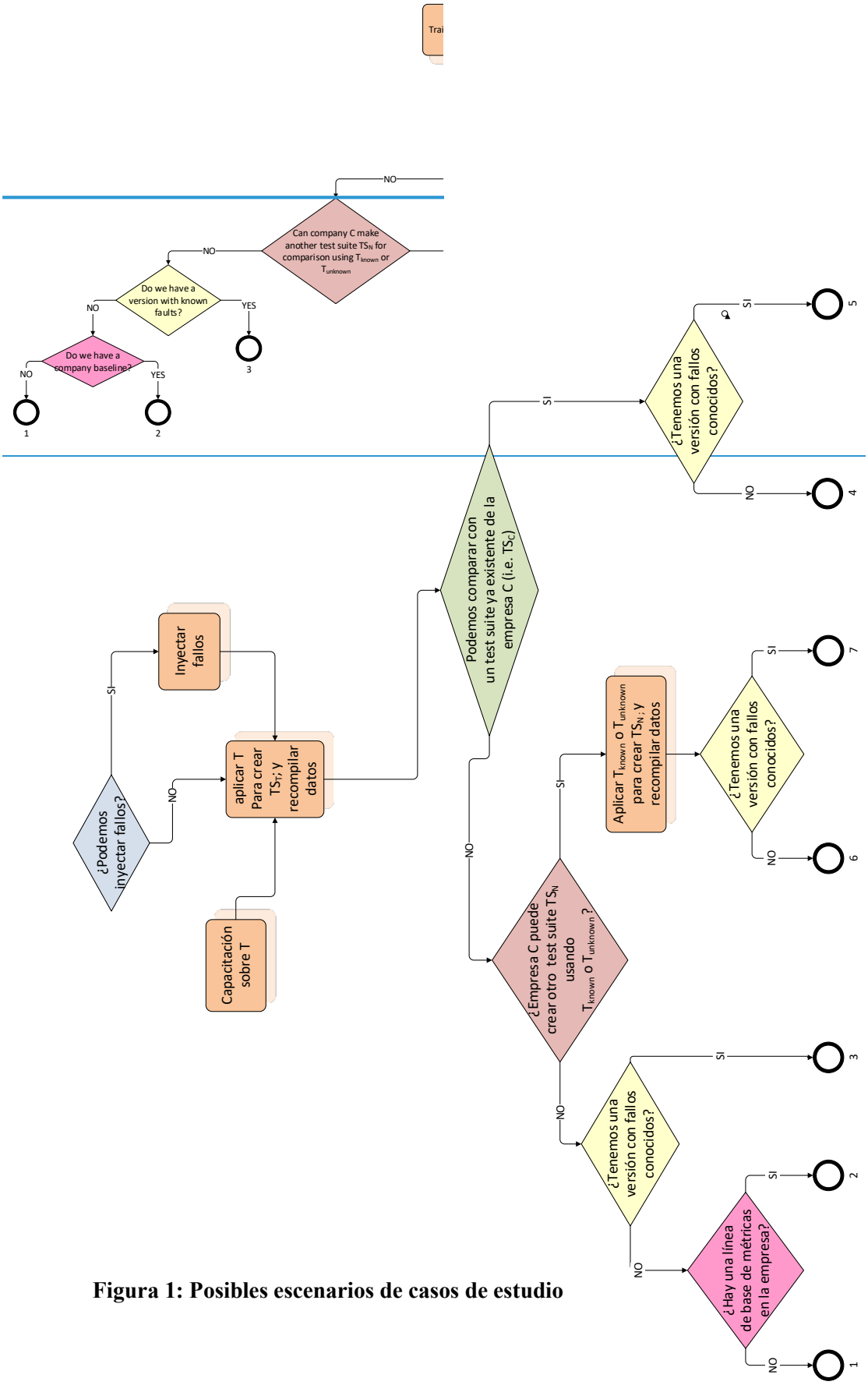

Figura 1: Posibles escenarios de casos de estudio