Un Framework Aspectual para la Evaluación de Usabilidad Automática de Tareas *

Roberto Farias, Natalia Trejo y Sandra Casas

GISP - Instituto de Tecnología Aplicada
Universidad Nacional de la Patagonia Austral
fariasrobertoa@gmail.com,nbtrejo@gmail.com,sicasas@yahoo.com

Abstract. La programación orientada a aspectos permite automatizar varias actividades de la evaluación de la usabilidad, en forma dinámica, transparente para el usuario y separada del código de las aplicaciones. A diferencia de otras propuestas, que se han centrado en pruebas que recolectan datos de bajo nivel de abstracción, este trabajo presenta un framework que permite realizar la evaluación de la usabilidad en aplicaciones de escritorio en el contexto de tareas de usuario. El framework consiste en un conjunto de módulos, aspectos y clases, que interactúan y se relacionan de manera tal de ser altamente reusable y sus requerimientos de especialización (configuración) son mínimos.

keywords: usabilidad, framework, programación orientada a aspectos, AspectJ

1 Introducción

La evaluación de usabilidad [1] automática refiere a la posibilidad de disponer de medios basados en software para realizar alguna de las actividades en las que se descompone (captura de datos, análisis o crítica) [6]. En lo que respecta a la captura de datos automatizada se ha caracterizado por usar archivos de logs. Sus ventajas son varias: soportan la ejecución de las pruebas en laboratorios de usabilidad o en el ambiente natural del usuario; son transparentes para los usuarios; permiten con facilidad obtener información de múltiples usuarios; permiten detectar errores comunes y actuales; etc. Los logs aportan gran cantidad de datos a los evaluadores de usabilidad, además de posibilitar la contrastación del comportamiento entre diferentes usuarios. Sin embargo, tienen desventajas, los logs generados por servidores webs o sistemas operativos, generan una gran cantidad de datos que resulta irrelevante para la evaluación de usabilidad, y son incapaces de registrar información específica. Por otro lado, añadir en las aplicaciones el código que realiza el logging de las actividades y tareas de usuario que interesan al evaluador de usabilidad, implica invadir la estructura interna de las aplicaciones agregando cientos de líneas de código, cuya remoción posterior es muy compleja.

* Este trabajo fue apoyado por el proyecto PIA315/1 y parcialmente financiado por UNPA, Sta. Cruz, Arg.

Las técnicas de separación de concerns [8], y en particular de la Programación Orientada a Aspectos (AOP) [9], proporcionan mecanismos para diseñar e implementar módulos de código que realicen la traza y el logging sin invadir y/o alterar la estructura interna de los sistemas. La posibilidad de realizar la recolección de los datos en forma dinámica, transparente para el usuario y separada del código de las aplicaciones a evaluar, ha provocado interés y varios enfoques se han presentado que aplican AOP a la evaluación de usabilidad. En particular aquellos propuestos para aplicaciones de escritorio (WIMP) ofrecen esquemas para la evaluación a bajo nivel. Posibilitan la traza de los elementos o eventos de interfaces y/o errores pero sin un contexto de significado (tarea) para el evaluador. En consecuencia carecen de una relevancia real a la hora de interpretar los resultados y/o calcular métricas, y proporcionar información útil para emitir una opinión al evaluador de usabilidad.

Luego de analizar los diseños presentados [14,11,4,12,13,2,5], se puede concluir que la razón radica en el hecho de que el concepto de "tarea" no es común en los vocabularios que participan de un escenario de evaluación de usabilidad y por ende no está presente en ambos contextos. En los sistemas software se representan los conceptos del dominio y lógica de negocios tales como: Cuenta, Cliente, Factura, Impuesto, Stock, etc., mientras que los evaluadores de usabilidad buscan identificar, analizar e interpretar tareas de usuario. En los sistemas software las "tareas de usuario" resultan ser entidades intangibles e inexistentes. Es por ello que los módulos (aspectos) que se diseñan e implementan para llevar adelante en forma automática las pruebas de la usabilidad, se restringen a los conceptos (entidades) existentes (métodos / atributos y clases del dominio), perdiéndose la concepción de la tarea. Planteamos como hipótesis, que sí es posible colocar en el centro la tarea, como concepto y como entidad, y por medio de aspectos conectarla al dominio y a los servicios que se requieran para la evaluación, entonces podrá ser identificada y analizada. En consecuencia se obtendrán datos que puedan ser interpretados a mayor nivel semántico y se lograrán resultados de mayor nivel de abstracción.

En este trabajo presentamos un framework que permite realizar la evaluación de la usabilidad en aplicaciones de escritorio en el contexto de tareas de usuario. El framework consiste de un conjunto de módulos, aspectos y clases, que interactúan y se relación de manera tal de ser altamente reusable y sus requerimientos de especialización (configuración) son mínimos. Este trabajo se estructura de la siguiente manera, en la Sección 2 se describe brevemente el modelo de usabilidad soportado por el framework, en la Sección 3 se presenta el Framework, en la Sección 4 se exponen algunos casos de estudio y finalmente las conclusiones y trabajo futuro.

2 Evaluación de Usabilidad

El proceso de evaluación implica varias actividades en función del método empleado, pero en general se agrupan en 3 pasos básicos: Captura de datos, Análisis y Crítica. Desde las propuestas de Nielsen [9] hasta la actualidad, se han planteado

una extensa y diversa cantidad de modelos y métricas para la evaluación de la usabilidad, por lo que en este trabajo no se proponen nuevas métricas. En su lugar se adoptará una propuesta que sirva al framework, que sea un modelo simple pero que puede escalar y sobre todo que haga eje en el concepto de tarea. El modelo de alto-nivel propuesto por Sauro y Kindlund [10] además de cumplir con las dimensiones ISO/ANSI (eficiencia, efectividad y satisfacción), sintetiza en cuatro métricas estas dimensiones y permite derivar otros indicadores útiles.

La eficiencia se mide calculando el tiempo de la tarea, la efectividad se mide contabilizando el total de errores producidos durante la ejecución de la tarea y si ésta se ha completado. Por último, la satisfacción es un valor promedio obtenido de 3 ítems de valoración subjetivos, basados en una puntuación (1 a 5) que representan la complejidad de la tarea para el usuario, la satisfacción y la consideración del tiempo que le ha requerido dicha tareas. La propuesta de Sauro no incluye automatización del modelo y las pruebas se realizaron en laboratorios controlados en períodos de tiempo de dos años.

3 Framework Aspectual

Los frameworks [7] pueden reducir los costos de desarrollo ya que permiten a los desarrolladores reusar experiencias previas en la resolución de problemas en los niveles de diseño e implementación del software. Investigaciones anteriores han demostrado que se pueden lograr altos niveles de reutilización del software a través del uso de frameworks ya que capturan aspectos comunes de una familia de aplicaciones. El esquema de la Figura 1 ilustra de manera general la aplicación del framework desarrollado. El objetivo del framework es proveer mecanismos que posibiliten la evaluación de usabilidad automática aplicando el modelo presentado en la Sección 2. Para diferentes aplicaciones de escritorio se proporciona un conjunto de módulos (clases-aspectos) que mediante la especialización de ciertos aspectos abstractos permiten reusar el diseño e implementación para calcular métricas y obtener información de la eficiencia, efectividad y satisfacción de las tareas de usuario.

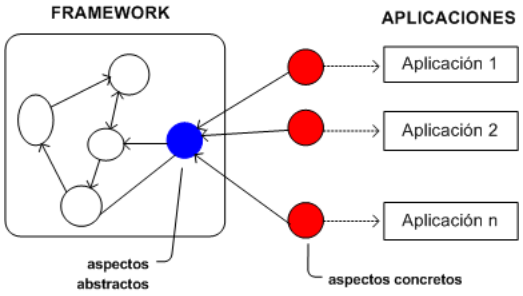

Fig. 1. Esquema general del framework.

3.1 Requerimientos

El framework ha sido construido desde cero aplicando un proceso iterativo de sucesivos refinamientos. Inicialmente se estableció el concepto y alcance de tarea. Definitivamente la tarea es un conjunto de acciones significativas para el evaluador que sucede durante la ejecución de la aplicación. Sin embargo, este concepto no está representado como entidad del sistema, por ser un concepto subyacente y subjetivo. La tarea entonces debe ser representada en el framework, resultando esta abstracción de una relevancia central dado que todo el framework requiere mantener relaciones e interacciones que son dependientes de la misma. Desde esta visión una tarea es una entidad subyacente a la aplicación que requiere encapsular un conjunto de estados y comportamiento interno, puesto que realiza una o más operaciones y provee un comportamiento global para lograr el objetivo específico. Por lo cual, cualquier y toda tarea podrá ser relacionada con algún evento del sistema que establece su inicio y también con otro evento que indica su finalización. Una tarea podrá ser instanciada diversas veces en la misma o distintas ejecuciones de la aplicación (sesiones), cada instancia deberá ser identificable, ya que comporta diferentes valores para los estados descriptores.

A partir del planteo de un escenario general y abstracto se ha procedido a identificar el conjunto de eventos que desencadenan las acciones que ejecutará y/o controlará el framework y el tipo de módulo (clase o aspecto) del framework que será responsable de ejecutar la función correspondiente (Tabla 1). Los eventos se clasifican en dos categorías según su origen. Los eventos de la aplicación (se ha producido un error mientras se ejecutaba la tarea) o los eventos pueden ser activados por el mismo framework (se instancia una tarea). Asimismo un evento puede desencadenar una o más acciones, las cuales pueden ser llevadas a cabo por distintos módulos.

Table 1. Eventos-Acciones-Módulos

Evento		Acciones/Responsabilidad		Módulo
E1	Ejecución del inicio de una tarea (D)	A1	Interceptar inicio de una tarea	Aspecto
		A2	Instanciar una tarea	
E2	Instanciar Tarea (F)	A3	Registrar inicio de la tarea	Clase
		A4	Logging inicio de la tarea	Aspecto
E3	Ejecución del fin de una tarea (D)	A5	Interceptar el fin de una tarea	Aspecto
		A6	Ordenar el fin de la tarea	
E4	Se produce un error (D)	A7	Identificar el error	Aspecto
		A8	Logging errores	
		A9	Contabilizar error	Clase
E5	Finalizar la tarea (F)	A10	Registrar fin de la tarea	Clase
		A11	Proporcionar el Cuestionario Satisfacción	Aspecto
		A12	Logging Datos del fin de la tarea	

Los eventos E1 y E3, originados en el dominio (D), permiten identificar, delimitar y hacer el seguimiento de la tarea durante su tiempo de vida (ejecución),

lo que implica que debidamente generarán las acciones de instanciación (A1-A2) y finalización(A5-A6), estas acciones serán llevadas a cabo por un aspecto. Al instanciar una tarea (E2) se comienzan a registrar los datos (A3) que serán necesarios para el cálculo de métricas (duración) y accionar el log de la tarea (A4). En esta sucesión de eventos y acciones, el framework condiciona una cantidad de acciones a eventos internos que están supeditados a los cambios de la tarea. Esta premisa hace a los módulos del framework más independientes del dominio.

3.2 Diseño e Implementación

En la Figura 2 se presenta el diseño del framework, la implementación está desarrollada en Java y AspectJ. La clase Task constituye el núcleo del framework, dado que mantiene diversas relaciones con los restantes módulos. Los aspectos tienen un rol preponderante, ya que son necesarios para conectar la tarea con la aplicación y para comunicarla con otros servicios del framework (logging). El aspecto TaskConnect conecta a la tarea con los correspondientes eventos de inicio y finalización y realiza las acciones de crear la instancia de la Task y ordenar su finalización. El aspecto TaskError intercepta los errores que se producen en la ejecución de la tarea creada por TaskConnect y el aspecto TaskSatisfaction facilita un formulario que permite registrar la información subjetiva cuando la tarea ha sido finalizada. El aspecto TaskLog registra el logging de las tareas, cuando estas se crean, finalizan o generan otros eventos, de manera que es dependiente de la tarea y no de la aplicación.

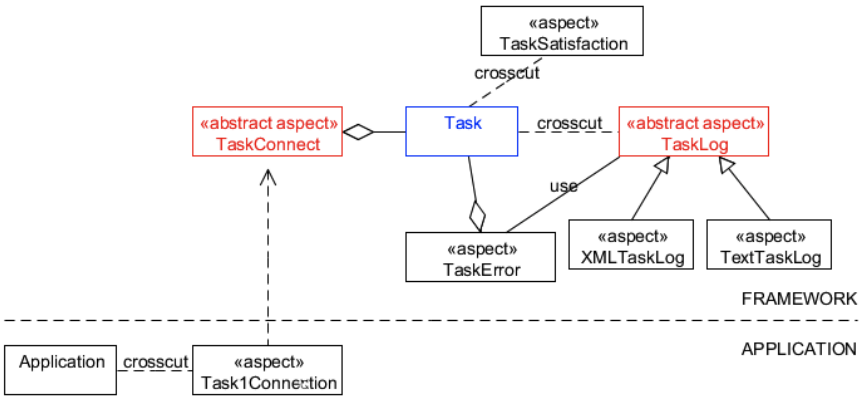

Fig. 2. Diseño del framework.

A continuación se presentan los principales detalles del diseño interno e implementación de los módulos del framework. Cómo se mencionó la clase Task representa una tarea del usuario. Tiene diversos estados que permiten calcular posteriormente métricas o que van acumulando los valores de las mismas. Estos estados son inicializados en su instanciación. El atributo init se inicializa con

la hora del sistema y el atributo complete con el valor falso. Estos dos atributos serán posteriormente actualizados por el método finalize (cuando la tarea finaliza) y usados para calcular el tiempo (eficiencia de la tarea) y si se ha completado (efectividad de la tarea). Para cada atributo que permite contabilizar errores como los atributos relacionados a la satisfacción existen métodos que permiten su correcta actualización.

```
class Task  {
  private String id,
  private Time init, end;
  private boolean complete;
  private int #exception, sat1, sat2, sat3;
  Task(..){
     // inicialización de estados}
   ...
  finalize()
    {// actualización de estados }
}
```

El aspecto abstracto TaskConnect conecta la tarea con la aplicación. Dos pointcuts abstractos startTask y endTask son necesarios para definir los joinpoints (clase y método del dominio) en los cuales comienza y finaliza cada tarea. Estos deberán ser definidos en un aspecto concreto por el desarrollador, que deberá crear un subaspecto por cada tarea de usuario distinta que pretenda evaluar. TaskConnect tiene por atributo una tarea, (objeto de tipo Task) que es instanciado por dicho aspecto cuando el pointcut startTask es activado y se ejecuta el aviso asociado, lo cual ocurre antes de que cada tarea inicie. De manera similar, este aspecto ordena la finalización de la tarea cuando luego que se activa el pointcut endTask, por medio del aviso asociado, se invoca al método.

```
abstract aspect TaskConnect {
   private String idTask;
   private Task t;
   abstract pointcut starTask;
   abstract pointcut endTask;
   before() : initTask();
   {  this.setTask();
      t = new Task(idTask);}
   after() : endTask()
   { t.finalize(); }
}
```

El aspecto TaskError tiene por objeto interceptar los errores que se producen durante la ejecución de una tarea y ordenar la actualización de los acumuladores correspondientes de la tarea. El aspecto mantiene una referencia a la tarea instanciada, por medio del pointcut beginning y el atributo monitored_task. Luego por cada tipo de error que se registra, se define un pointcut y advice. Tal es el caso del pointcut exception, cuyo objetivo es interceptar las excepciones que se producen mientras la tarea se ejecuta. Para ello es necesario especificar varias condiciones, que permiten delimitar al conjunto de excepciones que se producen.

Estas restricciones debido a que también son necesarias para la intercepción de otros tipos errores se especifican en forma aislada e individual en los pointcuts condition_1, condition_2 y condition_3. Luego en el pointcut exception se combinan con el descriptor de pointcut más específico, para este caso, handler. El advice asociado, primero actualiza el contador de excepciones de la tarea y luego ordena el log del error, operación que realiza el aspecto TaskLog.

```
aspect TaskError {
  Task monitored_task=null;
  pointcut inicialization():
              initialization(Task.new(String));
  pointcut beginning(Task t):inicialization()&&this(t);
  after(Task t): beginning(t)
    {monitored_task=t;}
  pointcut condition_1():!cflow(adviceexecution)
  pointcut condition_2():!cflow(inicialization)
  pointcut condition_3():if ((monitored_task!=null)&&
                            (!monitored_task.isComplete()))
  pointcut exception(Throwable e):
               handler(Throwable+) && args(e)
               && condition_1() && condition_2()
               && condition_3()
  before(Throwable e): exception(e) {
     // actualizar monitored_task;
     TaskLog.aspectof().logException(e);
  }
}
```

Cuando se produce la finalización de una tarea, (el método finalize de Task es invocado por el aspecto TaskConnect), el aspecto TaskSatisfaction proporciona un cuestionario que permite obtener información subjetiva del usuario relacionada a la dimensión satisfacción. Este formulario es muy simple, permite al usuario elegir un valor entre 1 y 5 para cada una de las 3 preguntas que se muestran en pantalla.

```
aspect TaskSatisfaction {
   pointcut satisfaction(Task t):
        execution(void Task.finalize(..))&&this(t);
   before(Task t): satisfaction (t) {
// desplegar cuestionario
       // obtener datos ingresados por el usuario.
// actualizar atrubutos sat1..de t
   return;}
}
```

Finalmente el aspecto TaskLog realiza el logging de la tarea en diversos momentos. Es una aspecto abstracto debido a que el log puede realizarse en diversos formatos (texto / xml / base de datos / consola). Los pointcut están definidos de manera concreta, ya que este aspecto está supeditado por completo a los cambios en la tarea.

```
abstract aspect TaskLog {
   abstract void logException() // log de excepciones
   abstract void logTask()
   pointcut logSart(Task): call Task.new(..) && target(t);
   pointcut logEnd(Task): call Task.finalize(..) && target(t);
   after() : logStart()
   { logTask(t);}
   after() : logEnd()
   { logTask(t);}
}
```

El diseño e implementación de los módulos del framework se ha apoyado en distintos patrones de diseño aspectuales [3]. Por ejemplo, en el caso de TaskConnect se aplicaron los patrones Abstract Pointcut y Template Advice, en TaskError se aplicó el patrón Composite Pointcut y en TaskLog se aplicó el patrón Template Advice.

4 Casos de Estudio

El desarrollo iterativo del framework implicó la ejecución de pruebas con cada versión del framework obtenida. El mismo ha sido probado y validado empleando dos aplicaciones de escritorio: JMoney (http://jmoney.sourceforge.net/) y Freemind (http://freemind.sourceforge.net/). Para cada una se crearon diversas tareas y en cada iteración se configuró el framework. Los principales objetivos de las pruebas fueron constatar que el registro de los datos y cálculo de las métricas funcionaba correctamente; y comprobar el nivel de reutilización, configuración y especialización. A continuación se presenta una tarea para cada caso de estudio, el aspecto que fue adaptado y el resultado de las pruebas (Fig. 3 y Fig. 4).

Freemind Tarea: Crear Mapa Mental Básico.

1. Crear un nuevo mapa, haciendo clic en el botón "Nuevo" desde la barra de herramientas, o desde el menú Archivo.
2. Completar el texto del nodo raíz del mapa creado.
3. Construir una jerarquía de tres niveles con nodos hijos y hermanos (al menos 11).Para ello se deberá posicionar en el nodo sobre el cual quiere crear un nodo hijo o hermano y luego insertar el nodo desde las opciones del menú contextual, o desde el menú Insertar.
4. Ir al menú herramientas y ordenar los nodos por nombre.
5. Guardar el mapa mental en el escritorio con un nombre significativo.

```
public aspect FreemindTask1Connect extends TaskConnect{
 private String idTask="Crear Mapa Conceptual Básico";
 pointcut startTask():execution(void
 freemind.modes.common.actions.NewMapAction. \\
 actionPerformed(ActionEvent));
 pointcut endTask():execution(
  *freemind.modes.ControllerAdapter.SaveAsAction.*(..)) ||
```

```
        execution(* freemind.modes.ControllerAdapter.\\
        SaveAction.*(..));
    }
```

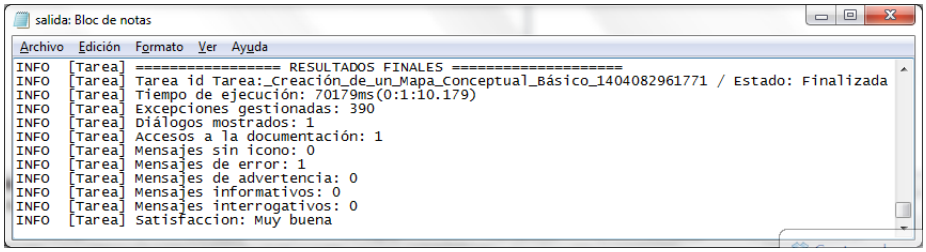

Fig. 3. Log de Tarea: Crear Mapa Conceptual Básico.

JMoney Tarea: Crear una nueva cuenta

1. Hacer click en el botón derecho del panel lateral y seleccionar la opción "new account".
2. Ingresar las propiedades de la cuenta desde el panel properties de la cuenta
3. Ingresar entradas en la cuenta desde el panel entries
4. Salvar desde el botón "save" en la barra de herramientas o desde menú "file/save", usando combinación de teclas "ctrl+s" o respondiendo afirmativamente al mensaje de grabar los cambios antes de cerrar la aplicación.

```
public aspect JMoneyTask1Connect extends TaskConnect{
     private String idTask="Crear Nueva Nuenta";
     pointcut starTask():execution(void
                  net.sf.jmoney.gui.MainFrame.newAccount());
     pointcut endTask():execution(void
               net.sf.jmoney.gui.MainFrame.saveSession()) ||
       execution(void net.sf.jmoney.gui.MainFrame.saveSessionAs());
}
```

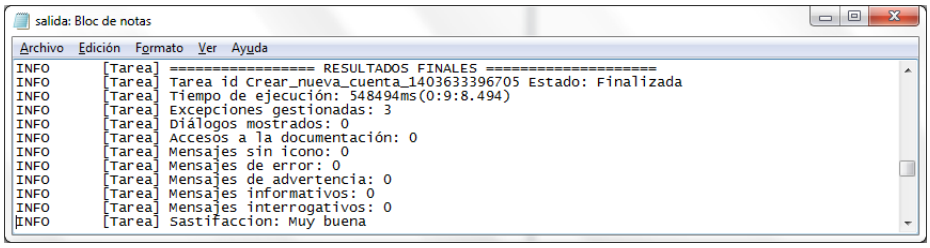

Fig. 4. Log de Tarea: Crear Nueva Cuenta.

5 Conclusiones

En este trabajo se ha presentado un framework que permite la evaluación de la usabilidad de aplicaciones de escritorio de alto nivel, dado que permite obtener información y calcular métricas a nivel de tareas de usuario, en las dimensiones de eficiencia, efectividad y satisfacción. El framework automatiza la recolección de datos y algunas funciones de análisis. Aunque el framework se compone de varios módulos, el desarrollador sólo debe configurar dos pointcut de un solo aspecto (TaskConnect), para que toda la evaluación de usabilidad de una determinada tarea sea efectuada. Lo cual implica un alto nivel de reutilización, del diseño y la implementación, ya que se ha logrado reducir al mínimo las condiciones de variación del problema y requerimientos de configuración. Los trabajos futuros refieren a incorporar más funcionalidad para la registración de otros datos relacionados a la tarea con el objeto de calcular nuevas métricas. También resulta de utilidad incorporar un visor que facilite la tarea de análisis y crítica posterior al evaluador de usabilidad. Completadas estas mejoras, el paso siguiente es desarrollar y evaluar un framework para aplicaciones web.

References

1. ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs) - Part 11 : Guidance on usability (1998)
2. Bateman, S., Gutwin, C., Osgood, N., McCalla, G.: Interactive Usability Instrumentation. In: 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems. pp. 45–54. ACM, New York, NY, USA (2009)
3. Hanenberg, S., Schmidmeier, A.: Idioms for Building Software Frameworks in AspectJ. In: 2nd AOSD Workshop on Aspects, Components and Patterns for Infraestructure Software. pp. 80–89. ACM, New York, NY, USA (2003)
4. Holzinger, A., Brugger, M., Slany, W.: Applying Aspect Oriented Programming in Usability Engineering Processes: On the Example of Tracking Usage Information for Remote Usability Testing. In: International Conference on Electronic Business and Telecommunications. pp. 53–56. SciTePress, Seville, Spain (2011)
5. Humayoun, S.R., Dubinsky, Y., Catarci, T.: UEMan: A Tool to Manage User Evaluation in Development Environments. In: ICSE'09. pp. 551–554. IEEE, Washington, DC, USA (2009)
6. Ivory, M.Y., Hearst, M.A.: The State of the Art in Automating Usability Evaluation of User Interfaces. ACM Comput. Surv. 33(4), 470–516 (2001)
7. Johnson, R.E.: Frameworks = (Components + Patterns). Communications of ACM 40(10), 39–42 (1997)
8. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-Oriented Programming. In: Akşit, M., Matsuoka, S. (eds.) ECOOP'97, LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
9. Nielsen, J.: The Usability Engineering Life Cycle. Computer 25(3), 12–22 (1992)
10. Sauro, J., Kindlund, E.: A Method to Standardize Usability Metrics into a Single Score. In: SIGCHI Conference on Human Factors in Computing Systems. pp. 401–409. ACM, New York, NY, USA (2005)

11. Shekh, S., Tyerman, S.: An Aspect-Oriented Framework for Event Capture and Usability Evaluation. In: Maciaszek, L., González-Pérez, C., Jablonski, S. (eds.) Evaluation of Novel Approaches to Software Engineering, CCIS'10, vol. 69, pp. 107–119. Springer Berlin Heidelberg (2010)
12. Tao, Y.: Automated Data Collection for Usability Evaluation in Early Stages of Application Development. In: ACACOS'08. pp. 135–140. WSEAS Press, Hangzhou, China (2008)
13. Tao, Y.: Aspect-Oriented Instrumentation for Capturing Task-Based Event Traces. International Journal on Control System and Instrumentation 3(1), 32–35 (2012)
14. Tarta, A.M., Moldovan, G.: Automatic Usability Evaluation Using AOP. In: IEEE International Conference on Automation, Quality and Testing, Robotics. vol. 2, pp. 84–89. IEEE, Cluj-Napoca, Romania (2006)