TFG: Análisis Emocional para la Inclusión Digital

Gema Eugercios Suárez
Paloma Gutiérrez Merino
Elena Kaloyanova Popova

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Mayo 2018

Documento maquetado con TEXIS v.1.0.

Este documento está preparado para ser imprimido a doble cara.

TFG: Análisis Emocional para la Inclusión Digital

Informe técnico del departamento
Ingeniería del Software e Inteligencia Artificial
IT/2009/3

Versión 1.0

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Mayo 2018

ISBN 978-84-692-7109-4

Al duque de Béjar
y
a tí, lector carísimo

I can't go to a restaurant and order food because I keep looking at the fonts on the menu.
Donald Knuth

Agradecimientos

A todos los que la presente vieren y entendieren.

Inicio de las Leyes Orgánicas. Juan Carlos I

Groucho Marx decía que encontraba a la televisión muy educativa porque cada vez que alguien la encendía, él se iba a otra habitación a leer un libro. Utilizando un esquema similar, nosotros queremos agradecer al Word de Microsoft el habernos forzado a utilizar LaTeX. Cualquiera que haya intentado escribir un documento de más de 150 páginas con esta aplicación entenderá a qué nos referimos. Y lo decimos porque nuestra andadura con LaTeX comenzó, precisamente, después de escribir un documento de algo más de 200 páginas. Una vez terminado decidimos que nunca más pasaríamos por ahí. Y entonces caímos en LaTeX.

Es muy posible que hubíeramos llegado al mismo sitio de todas formas, ya que en el mundo académico a la hora de escribir artículos y contribuciones a congresos lo más extendido es LaTeX. Sin embargo, también es cierto que cuando intentas escribir un documento grande en LaTeX por tu cuenta y riesgo sin un enlace del tipo "*Author instructions*", se hace cuesta arriba, pues uno no sabe por donde empezar.

Y ahí es donde debemos agradecer tanto a Pablo Gervás como a Miguel Palomino su ayuda. El primero nos ofreció el código fuente de una programación docente que había hecho unos años atrás y que nos sirvió de inspiración (por ejemplo, el fichero `guionado.tex` de TeXiS tiene una estructura casi exacta a la suya e incluso puede que el nombre sea el mismo). El segundo nos dejó husmear en el código fuente de su propia tesis donde, además de otras cosas más interesantes pero menos curiosas, descubrimos que aún hay gente que escribe los acentos españoles con el `\'{\i}`.

No podemos tampoco olvidar a los numerosos autores de los libros y tutoriales de LaTeX que no sólo permiten descargar esos manuales sin coste adicional, sino que también dejan disponible el código fuente. Estamos pensando en Tobias Oetiker, Hubert Partl, Irene Hyna y Elisabeth Schlegl, autores del famoso "The Not So Short Introduction to LaTeX 2_ε" y en Tomás

Bautista, autor de la traducción al español. De ellos es, entre otras muchas cosas, el entorno `example` utilizado en algunos momentos en este manual.

También estamos en deuda con Joaquín Ataz López, autor del libro "Creación de ficheros LaTeX con GNU Emacs". Gracias a él dejamos de lado a WinEdt y a Kile, los editores que por entonces utilizábamos en entornos Windows y Linux respectivamente, y nos pasamos a emacs. El tiempo de escritura que nos ahorramos por no mover las manos del teclado para desplazar el cursor o por no tener que escribir `\emph` una y otra vez se lo debemos a él; nuestro ocio y vida social se lo agradecen.

Por último, gracias a toda esa gente creadora de manuales, tutoriales, documentación de paquetes o respuestas en foros que hemos utilizado y seguiremos utilizando en nuestro quehacer como usuarios de LaTeX. Sabéis un montón.

Y para terminar, a Donal Knuth, Leslie Lamport y todos los que hacen y han hecho posible que hoy puedas estar leyendo estas líneas.

Resumen

...

...

...

Índice

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción

...

...

Resumen: Este capítulo sirve como introducción al trabajo que se va a realizar.

1.1. Introducción

...

Notas bibliográficas

Citamos algo para que aparezca en la bibliografía... (Bautista et al., 1998)

Y también ponemos el acrónimo CVS para que no cruja.

Ten en cuenta que si no quieres acrónimos (o no quieres que te falle la compilación en "release" mientras no tengas ninguno) basta con que no definas la constante `\acronimosEnRelease` (en `config.tex`).

En el próximo capítulo...

...

Capítulo 2

Estado del arte

...

...

Resumen: En este capítulo se van a tratar los aspectos más importantes tanto de la computación emocional como de las diferentes tecnologías y metodologías que se van a utilizar. En primer lugar, en la sección 2.1, se define la computación emocional y sus posibles aplicaciones, también se explican los distintos diccionarios afectivos ya existentes que permiten la marcación emocional de textos y se presenta el diccionario que se va a utilizar en este trabajo. En la sección 2.2 se introduce la tecnología que se va a utilizar para implementar el trabajo, los Servicios Web. En la sección 2.3 se fijan los conceptos relacionados con la metodología Scrum. En la sección 2.4 se explican las bases de la integración continua aplicada al desarrollo de Software y cómo se va a aplicar en este trabajo.

2.1. Computación Afectiva

La computación emocional es el estudio y el desarrollo de sistemas y dispositivos capaces de percibir, medir e interpretar las emociones humanas.Esta rama de la computación permite un avance notable en la inteligencia artificial, hasta tal punto que los ordenadores lleguen a adaptarse a los humanos, sus necesidades y estados de ánimo. Los seres humanos están rodeados de emociones, en cualquier ámbito de su vida, tanto de las suyas propias como las de las personas con las que se comunican. Tanta importancia tienen para nosotros que influye no sólo a nuestra comunicación, sino también en nuestro aprendizaje y toma de decisiones. Por ello, resulta artificial y en ocasiones incluso frustrante intentar conmunicarse con una máquina que no es capaz de computar sentimientos. Se acuña entonces el

término computación afectiva (Picard,1997) pretende mejorar la interacción hombre-máquina haciéndola más natural y asequible.

El funcionamiento de este tipo de sistemas se basa en identificar el estado emocional del sujeto a través de diferentes fuentes(voz, expresiones, señales fisiológicas, palabras...) y procesar la información para clasificarla y aprender de ella. Clasificar la información de entrada puede resultar complicado ya que se suelen recibir varias señales diferentes a la vez, lo que hace necesario utilizar técnicas de priorización para determinar cuáles son las que más aportan a la hora de analizar y gestionar la información. Una vez identificada la emoción predominante, el sistema responde adecuandose a ella. La salida dependerá del tipo de sistema y las herramientas de las que dispone este para expresar su respuesta (colores, sonidos, emoticonos...) En casos más complejos aplicados a robótica o modelado, la salida producida es una simulación de la respuesta que produciría un ser humano ante los estímulos recibidos imitando su expresión corporal, voz o gestos faciales.

Se trata de una tecnología con un espectro enorme de posibilidades, ya que como se ha mencionado antes, las emociones están presentes en todos los ámbitos de la vida de una persona. Por lo tanto, puede aplicarse a áreas muy diferentes entre sí. Algunas de estas áreas son:

- **Marketing**: Actualmente una de sus aplicaciones más explotadas. Poder evaluar la reacción emocional de alguien ante un anuncio o producto es una estrategia comercial que ya está siendo utilizada por cada vez más empresas.
- **Salud**: Principalmente detección del estrés para minimizar sus efectos y aprender a controlarlo. Es posible inferir el nivel de estrés de una persona midiendo sus señales fisiológicas (ritmo cardiaco, respiración...) y si este nivel es demasiado alto proceder en consecuencia según el tipo de sistema. Se puede aplicar de forma similar a las fobias.
- **Entretenimiento**: La industria de los videojuegos ha crecido mucho en los últimos años e introducir este tipo de tecnología permite a las compañías crear juegos más adaptables y cercanos al jugador, lo que atrae a más público y mejora la experiencia de juego (modelado de personajes más realista y mayor empatía con estos).
- **Robótica**: El mayor problema de los robots diseñados para interactuar con humanos es la carencia de emociones. Algunos de ellos llegan a producir una sensación de incomodidad. Dotar a este tipo de robots de cierta "humanidad"no sólo haría más cómodo el tratar con ellos sino que podrían realizar tareas como el acompañamiento de personas mayores.
- **Accesibilidad**: Las herramientas que puede generar la computación afectiva pueden servir de gran ayuda a personas que no tengan la capa-

cidad de entender o expresar sus emociones, como por ejemplo personas con autismo. Para este tipo de personas sería extremádamente complicado acceder a la tecnología sin la inclusión de estas herramientas que sean capaces de adaptarse a sus necesidades.

En este trabajo vamos a centrarnos en el último área, la accesibilidad. En particular, en facilitar a personas que padecen Trastornos del Espectro Autista (TEA) el entendimiento de textos. Trataremos de analizar un texto de entrada para identificar las emociones predominantes y en que medida se presentan, evitando las posibles ambigüedades que pueda haber. Las emociones con las que trataremos son las básicas: alegría, tristeza, miedo, sorpresa, enfado o neutral. Una vez identificadas las emociones se etiquetarán para hacerlas más explícitas mediante emoticonos.

Para deducir la emoción que transmite una palabra específica utilizaremos un diccionario afectivo basado en otros ya existentes.

2.1.1. Diccionarios Afectivos Existentes

Diccionarios que existen actualmente.

2.1.2. Nuestro diccionario

El diccionario que vamos a utilizar.

2.2. Servicios Web

Aquí va la parte de Servicios Web.

2.3. Metodología Scrum

2.3.1. Proceso de desarrollo software

Un proceso, es un conjunto de actividades, acciones y/o tareas que se realizan para crear un producto determinado; en nuestro caso un producto software.

El proceso de desarrollo software, como esctructura general, consta de cinco grandes actividades: **comunicación**, **planeación**, **modelado**, **construcción** y **despliegue**. Estas actividades, siempre se realizan en este orden; pero dependiendo del modelo que se emplee variará el flujo; algunos flujos son: lineal, iterativo, evolutivo o paralelo.

El desarrollo de software hace unos años era algo novedoso y tan moderno que no había estrategias claras para crear productos. Por lo que, con motivo de ordenar todo el proceso se crearon diferentes modelos de proceso. Un

modelo de proceso no es más que una estructura para realizar las actividades que forman un proceso.

Primeramente, se crean los modelos llamados tradicionales son procesos muy controlados y con muchas normas y políticas. Se le da mucha importancia a la arquitectura del software y se expresa mediante modelos. El cliente interactúa con el equipo de desarrollo mediante reuniones separadas en el tiempo. En la primera reunión se fija un contrato que el equipo debe seguir, lo que genera muchos problemas a la hora de desarrollar ya que el equipo tomará decisoines sin tener en cuenta lo que el cliente opinaría al respecto. Podríamos decir entonces que es un modelo impuesto externamente no solo por el contrato con el cliente sino porque está basado en normas de estándares. Estos modelos se llevan a cabo en proyectos con equipos grandes y distribuidos. Generan muchos artefactos y documentación.

Más adelante surgen los modelos de desarrollo ágiles. La agilidad en términos de software se podría definir como la respuesta efectiva al cambio; pero va más allá, esta idea lleva consigo toda una filosofía determinada en el Manifiesto por el Desarrollo Ágil. Este Manifiesto propone un nuevo modelo de proceso en el que los idividuos e interacciones están por encima de procesos y herramientas; el software funcionando sobre la documentación extensiva; la colaboración con el ciente sobre la negociación contractual y la respuesta ante el cambio sobre seguir un plan. También se han redactado doce principios:

1. *La mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.*

2. *Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrolo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.*

3. *Entregamos sotware funcional frecuentemetne, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.*

4. *Los responsables de negocio y los desarrolladores trabajan juntos de forma cotidiana durante todo el proyecto.*

5. *Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y apoyo que necesitan y confiarles la ejecución del trabajo.*

6. *El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.*

7. *El software funcionando es la medida principal de progreso.*

8. *Los procesos Ágiles promueven el desarrollo sostenible. Los pomotores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de forma indefinida.*

9. *La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.*

10. *La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.*

11. *Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.*

12. *A intervalos regulares el equipo reflexiona sobe cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.*

Dado que los modelos ágiles están diseñados para poder realizar cambios en los requisitos en cualquier momento y están pensados para pequeños grupos de trabajadores. Creemos muy conveniente para nuestro trabajo usar una metodología ágil.

2.3.2. Modelo ágil: SCRUM

SCRUM es una metodología ágil para gestionar el desarrollo de software. Fue definido por *Ikujiro Nonaka* e *Hirotaka Takeuchi* a principios de los 80; compararon la forma de trabajo con el avance en formación de scrum (melé en español) de los jugadores de rugby y por esta razón se llama la metodología SCRUM. *Nonaka* y *Takeuchi* caracterizan SCRUM por el protagonismo de equipos brillantes, auto-organizados y motivados que abordan el desarrollo de sistemas complejos partiendo de una visión general y solapando las fases del desarrollo. Más adelante, en 1995, *Ken Schwaber* presentó una metodología basada en un ambiente SCRUM y usó el mismo término para definir la metodología. Después, en 2005, *Mike Cohn*, *Esther Derby* y *Ken Schwaber* organizaron la «Scrum Alliance» para difundir el marco de trabajo para el desarrollo software basado en la metodología SCRUM.

SCRUM divide el trabajo en diferentes unidades llamadas *sprints*, estos tienen una duración preestablecida de entre dos y cuatro semanas obteniendo siempre al final una versión del software con nuevas prestaciones listas para ser usadas. En cada *sprint* se ajusta la funcionalidad y se añaden nuevas prestaciones priorizando aquellas que aporten más valor.

A continuación, explicaremos algunos conceptos que son necesarios para entender la metodología.

Esta metología, hace mucho énfasis en el «equipo de trabajo». Este equipo está formado por diferentes roles.

- **Product Owner**: representa al cliente. Este no está implicado directamente en el proyecto, pero se encarga de definir los objetivos y de garantizar que el equipo trabaja de manera adecuada. Tal y como hemos visto anteriormente, las metologías ágiles incluyen al cliente de una manera más cercana al desarrollo y SCRUM lo hace de esta manera.
- **Scrum Master**: es el encargado de asegurar que el equipo no tiene problemas en sus tareas. Ayuda y guía al Scrum Team. Diríamos que es el encargado de que todo el proyecto salga adelante.
- **Scrum Team**: Como su nombre indica, es el equipo encargado de desarrollar y entregar el producto.

También define una serie de artefactos, en menor medida que cualquier modelo tradicional.

- **Product backlog**: es una lista realizada por el usuario en la que explica los requisitos del producto, ordenados por prioridad. A lo largo del desarrollo crece y evoluciona. Se denomina tambien «historias de usuario».
- **Sprint backlog**: es una lista de las tareas que debe realizar el equipo durante el sprint. Podríamos decir que son las historias de usuario que el equipo decide realizar en un sprint.
- **Incremento**: es el resultado de cada sprint.
- **Burn down chart**: es una gráfica de avance que mide la cantidad de requisitos del backlog del proyecto pendientes al comienzo de cada sprint, esta es actualizada a diario para comprobar el avance.

Por último, define una serie de eventos.

- **Sprint**: este evento ya está explicado. Un sprint es cada una de las iteraciones del desarrollo.
- **Sprint planning**: es una reunión del equipo de trabajo donde se decide qué se va a realizar en un sprint y las tareas en las que se divide el objetivo.
- **Dialy sprint meeting**: es una reunión diaria como máximo de quince minutos, de pie, en donde cada componente del equipo informa sobre cómo va en sus tareas, lo que hizo el día anterior, lo que hará ese día y los problemas que ha encontrado o los que cree que se va a encontrar.
- **Sprint Review**: es una reunión que se realiza al concluir el sprint donde se analiza el incremento creado, las tareas realizadas, si se ha concluido todo a tiempo, impedimentos y problemas.

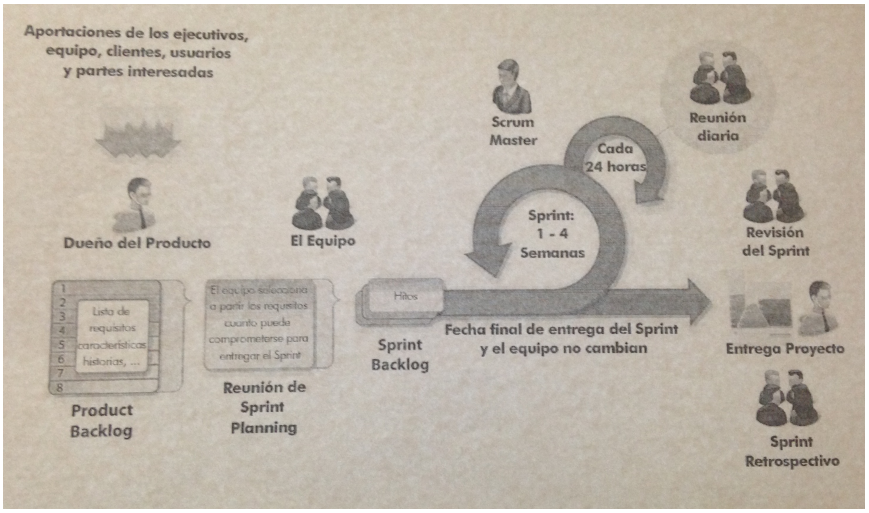

Figura 2.1: Flujo SCRUM

- **Sprint Retrospective**: es una reunión donde se habla de cómo ha funcionado el equipo en el sprint y qué cosas se pueden mejorar para el siguiente a nivel de proceso o metodología. En esta reunión participa todo el scrum team.

Conociendo todos estos conceptos, nos resultará más sencillo comprender el flujo de trabajo.

En la Figura 2.1 podemos ver un diagrama del flujo SCRUM. Sería el siguiente: el cliente crea su Product backlog con los requisitos y características por orden de prioridades. A continuación, en un Sprint planning, se presenta el product backlog y el equipo decide qué actividades van a desarrollar y cuánto tiempo van a tardar. Después de esta reunión se elabora el Sprint Backlog con todas las actividades que van a realizar divididas en tareas; cada componente del grupo se asigna una tarea y en cuanto acabe seguirá con la siguiente actividad que no esté hecha. Es muy importatne el orden de las tareas ya que, el cliente ha ordenado en el product backlog las actividades por prioridad y el equipo desarrollará estas siguiendo dicho parámetro. A continuación se comienza el sprint con el tiempo establecido. Este tiempo no se puede cambiar ni el equipo que está trabajando. Cada día, 24 horas, el equipo se reunirá en una Dialy sprint meeting para poner en común lo explicado anteriormente. Al final del sprint se realiza la Sprint Review donde se entrega el incremento o el producto hasta el momento al cliente; por lo tanto en esta reunión están presentes el scrum team, el scrum master y el cliente. Por úlitmo el scrum team junto con el scrum master se vuelve a reunir para hacer el Sprint Retrospective.

2.4. Integración Continua

La práctica de desarrollo de software que se va a utilizar en el desarrollo de este trabajo es la integración continua. Se basa en que los desarrolladores combinen todos los cambios que realicen en el código en un repositorio común de forma periódica, de tal forma que una vez subidos estos cambios, se ejecutan una serie de pruebas automáticas sobre estos con el fin de validarlos. Aparte de las pruebas automáticas se realizan pruebas manuales para buscar errores más difíciles de encontrar que requieran a alguien que piense. Hasta que las pruebas sobre el nuevo código no acaban no se puede continuar. Aplicando esto a la metodología Scrum que vamos a utilizar, una historia de usuario no se puede considerar acabada hasta que no pase todas las pruebas, tanto automáticas como manuales.

Implementar esta práctica al desarrollo tiene importantes ventajas entre las que se encuentran:

- **Detección de errores:** Su objetivo principal es detectar los errores lo antes posible y así solucionarlos según surgen. Cada vez que el código cambia se compila y somete a pruebas para garantizar que no hay bugs. Este proceso aumenta la calidad del software y minimiza los riesgos del proceso ya que se tiene control sobre las versiones en todo momento. Cuanto más se tarde en detectar un error más trabajo lleva arreglarlo.

- **Visibilidad del proceso:** Todos los pasos que se realizan en el desarrollo son visibles a todo el equipo, que tiene una estrategia común muy bien definida.

- **Mejora del equipo:** Los desarrolladores no solo tienen una visión muy clara y estructurada del proceso sino que también aprenden a realizar todo tipo de pruebas, lo que les hace mejorar a nivel profesional.

Lo primero para poder utilizar integración continua es tener definido un **«pipeline»**, es decir, un conjunto de fases por las que tiene que pasar el software y que están automatizadas. Se establecen criterios para que el código pase de una fase a otra y estrategias para gestionar errores que puedas surgir en las diferentes fases (control de versiones). Es importante tener bien definidas las pruebas que se van a realizar sobre cada fase y que estas puedan garantizar la máxima corrección posible sin tardar más de lo admisible, ya que se necesita un feedback rápido para poder seguir avanzando en el proceso. Cada fase es un grupo de pruebas y cada subida de código es un **«pipeline»** distinto que avanza de forma independiente por las fases. Por lo tanto se sabe en todo momento en qué punto se encuentra una versión específica. Esto permite tener una visión general de todo el proceso facilitando notablemente la detección de errores en fases y pipelines concretos.

Para el correcto funcionamiento de esta práctica tiene que haber pequeñas integraciones de forma frecuente, una vez al día por ejemplo. Cuantos menos cambios haya más fácil es la integración en el producto general y solucionar los posibles errores que esta pueda generar. Cabe destacar que aunque una parte de código funcione de forma independiente no implica que vaya a funcionar al integrarlo en un programa más grande, por ello cuanto más frecuentes sean las integraciones mejor.

En nuestro caso la integración continua se aplicará de la siguiente manera:

- **Repositorio:** Se utilizará un repositorio común de *GitHub* en el que se subirán todos los cambios realizados en el código. Al ser un equipo de desarrollo pequeño y estar utilizando la metodología Scrum en principio todos los miembros del equipo estarán trabajando en la misma rama. Esto puede llegar a bloquear el proceso mientras una versión acabe de pasar las pruebas.
- **Jenkins:** Tendremos Jenkins corriendo en un servidor para realizar las pruebas automáticas. Cada vez que se detecten cambios en el repositorio este avisará al servidor que procederá a hacer las pruebas de compilación y funcionamiento sobre el nuevo código.
- **Pruebas manuales:** Algún miembro del equipo realizará las tareas de «tester». Después de las pruebas automáticas se realizará otra serie de pruebas planificadas de antemano para probar mejor la funcionalidad del código. El «tester» no será siempre la misma persona.

Notas bibliográficas

Citamos algo para que aparezca en la bibliografía... (Bautista et al., 1998)

Y también ponemos el acrónimo CVS para que no cruja.

Ten en cuenta que si no quieres acrónimos (o no quieres que te falle la compilación en "release" mientras no tengas ninguno) basta con que no definas la constante `\acronimosEnRelease` (en `config.tex`).

En el próximo capítulo...

En el próximo capítulo se tratarán los primeros servicios web.

Apéndice A

Así se hizo...

...

...

Resumen: ...

A.1. Introducción

...

Bibliografía

Y así, del mucho leer y del poco dormir,
se le secó el celebro de manera que vino
a perder el juicio.

Miguel de Cervantes Saavedra

Bautista, T., Oetiker, T., Partl, H., Hyna, I. y Schlegl, E. *Una Descripción de LaTeX 2ε*. Versión electrónica, 1998.

–¿Qué te parece desto, Sancho? – Dijo Don Quijote –
Bien podrán los encantadores quitarme la ventura,
pero el esfuerzo y el ánimo, será imposible.

Segunda parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes

–Buena está – dijo Sancho –; fírmela vuestra merced.
–No es menester firmarla – dijo Don Quijote–,
sino solamente poner mi rúbrica.

Primera parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes