Qué es la Ingeniería del Software

Pablo Sánchez Barreiro

Dpto. Matemáticas, Estadística y Computación
p.sanchez@unican.es

Resumen

Este documento describe brevemente qué es la *Ingeniería del Software*, qué elementos comprende y cuales son sus objetivos. La finalidad de este documento es servir de guía y ayuda a los alumnos de la titulación de *Grado en Ingeniería Informática* en su no trivial decisión acerca de qué itinerario curricular escoger: *Ingeniería de Computadores*, *Computación* o *Ingeniería de Software*.

1. ¿Qué es la Ingeniería del Software?

El término *Ingeniería del Software* se definió por primera vez en Garmisch (Alemania), en una conferencia patrocinada por la OTAN (*Organización del Tratado Atlántico Norte*) [Naur and Randell, 1968], en la cual se pretendía abordar el difícil problema de construir software libre de errores bajo unas restricciones de tiempo y coste predecibles y asumibles. En dicha conferencia se empieza a gestar el concepto de Ingeniería del Software como la disciplina que *"estudia métodos rigurosos para diseñar y construir software que, con una cierta certeza, haga lo que se supone que debe hacer")* [acm, 2005].

Actualmente, la *Ingeniería del Software* se suele definir como *"la disciplina que estudia y trata acerca del desarrollo y mantenimiento eficiente, sistemático y costeable de sistemas software robustos que satisfacen los requisitos de los usuarios que los utilizan"* [Piattini et al., 2003, acm, 2005, Pressman, 2009, Sommerville, 2010]. La Ingeniería del Software comprende tanto aspectos técnicos del desarrollo de un producto software Bourque and Dupuis [2004], tales como técnicas para desarrollar software más fácilmente adaptable [Gamma et al., 1994], como aspectos de gestión y dirección de proyectos, tales como la elaboración de presupuestos [Boehm et al., 2000, Jones, 2007]. La Figura 1 muestras las distintas fases en las que normalmente se descompone un proyecto de desarrollo software.

Todo proceso de desarrollo de un sistema software, una vez que una compañía software ha decidido emprender dicha tarea, comienza con una fase inicial de *análisis y especificación de requisitos* [Cockburn, 2000, Hull

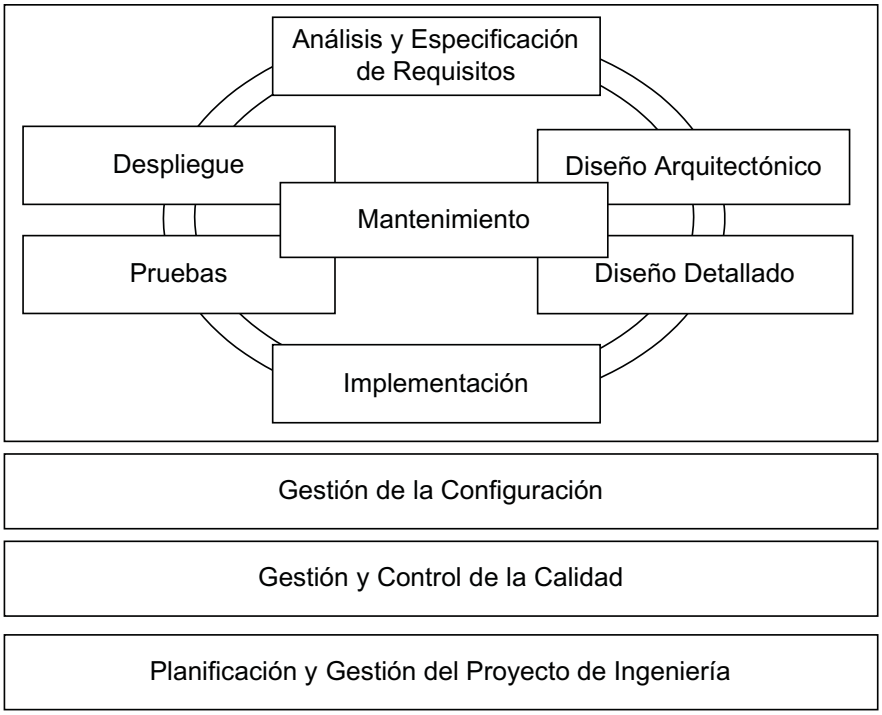

Figura 1: Esquema general de un proceso de desarrollo software

et al., 2011, Pohl, 2010]. El objetivo de dicha fase es determinar qué debe hacer el sistema software de acuerdo a las necesidades de los usuarios finales. Esta es una tarea bastante compleja pues en muchas ocasiones el *Ingeniero de Requisitos* debe adquirir un amplio conocimiento acerca del dominio para el cual se desarrolla la aplicación o familiarizarse con los procesos de trabajo de la empresa que va a adquirir el producto software. Es además una tarea crucial, porque si no entendemos bien qué es lo que debe hacer el sistema software, podemos acabar construyendo una aplicación técnicamente muy buena pero que no resulte de utilidad al usuario final, pues no resuelve el problema que le atañe.

Como resultado de la fase de *Ingeniería de Requisitos*, el *Ingeniero de Requisitos* produce una *especificación de requisitos*. Dicho documento describe *qué* debe hacer el sistema software, sin entrar en detalles técnicos de *cómo* debe hacerlo. El *Ingeniero de Requisitos* debe determinar también qué propiedades, tales como la eficiencia, la disponibilidad o la seguridad, son importantes para un correcto funcionamiento de la aplicación, así como cuán importantes son cada una de estas propiedades dentro del dominio concreto donde se va a desplegar la aplicación.

Este análisis es importante pues no todas las propiedades se pueden sa-

tisfacer de forma efectiva al mismo tiempo, dado que suelen existir conflictos entre ellas. Por ejemplo, la seguridad y el rendimiento suelen entrar en conflicto, pues cuantas más comprobaciones de seguridad realicemos, menor será en rendimiento de la aplicación. Y al revés, cuanto menos comprobaciones realicemos, mejor será el rendimiento. Por tanto, dependiendo de cuán importante sean ambas propiedades para un sistema dado, habrá que alcanzar un compromiso intermedio entre seguridad y rendimiento.

En la segunda fase del proceso de desarrollo de un sistema software, usualmente denominada *diseño arquitectónico* [Shaw and Garlan, 1996, Bosch, 2000, Bachmann et al., 2010, Bass et al., 2003, Taylor et al., 2009], los arquitectos software, partiendo de la especificación de requisitos creada en la fase anterior, dividen el sistema a desarrollar en sus principales partes constituyentes, denominadas *subsistemas* o *componentes*. El objetivo es dividir el sistema en unidades más o menos independientes de forma que dichas unidades puedan ser desarrolladas y probadas de forma independiente por equipos de desarrollo diferentes, obteniéndose el sistema final mediante la composición de estos componentes. Para ello es crucial especificar de forma precisa los puntos de comunicación o *interfaces* entre tales unidades de descomposición, estableciendo de forma clara qué servicios son proporcionados y requeridos por cada una de ellas.

Además, los arquitectos software deberán asegurar de que el sistema no sólo cumple con sus requisitos funcionales, sino que además satisfacen en mayor o menor medida las propiedades, tales como robustez o disponibilidad, deseadas por el usuario y contempladas en la especificación de requisitos.

El siguiente paso en el proceso de desarrollo de un producto software consiste en el *Diseño Detallado* de cada uno de los subsistemas o componentes que conforman la arquitectura de un sistema. Dependiendo de la naturaleza de dichos componentes, se deberán aplicar técnicas diferentes. Por ejemplo, no es lo mismo diseñar un componente que representa una interfaz gráfica de usuario, donde priman conceptos como la usabilidad [Scott and Neil, 2009, Tidwell, 2011], que un componente destinado al almacenamiento persistente de datos [Piattini et al., 2006, Ehlmann, 2009, Elsmari and Navathe, 2010], donde los factores más cruciales son la integridad y la seguridad de los datos [Afyouni, 2005, Gertz and Jajodia, 2010, del Peso et al., 2008, Pieprzyk et al., 2010].

Una vez diseñado cada componente, se procede a la fase de *implementación*. Dependiendo de las características de cada componente, su implementación puede ser más o menos directa. Por ejemplo, implementar una base de datos a partir de su modelo relacional es una tarea más o menos trivial, la cual se puede realizar normalmente mediante generadores de código u otras técnicas de programación generativa [Czarnecki and Eisenecker, 2000]. En otros casos, como el de software de comunicaciones con un comportamiento fuertemente dirigido por complejos eventos externos, dicha transición puede no ser tan trivial [Weigert and Dietz, 2003].

Además, para la tarea de implementación los desarrolladores suelen apoyarse en diferentes marcos de trabajo (*frameworks*), tales como *Struts* [Brown et al., 2008], plataformas *middleware*, tales como *CORBA (Common Object Request Broker Architecture)* [Balen, 2000, Aleksy et al., 2010] o Spring [Walls, 2011] o generadores de enlaces objeto-relacional (*ORM, Object-Relational Mapping*) [Roebuck, 2011], tales como Hibernate [Bauer and King, 2006, Elliott et al., 2008], de forma que se pueda generar y reutilizar tanto código como sea posible.

Una vez concluida la implementación de dichos componentes, se deben realizar las *pruebas* pertinentes para verificar el correcto funcionamiento de los mismos [Burnstein, 2003, Baker et al., 2007, Myers et al., 2011, Willcock et al., 2011]. Para ello, los ingenieros de pruebas diseñan conjuntos de casos de pruebas, o *tests*, que permitan comprobar que los componentes desarrollados funcionan correctamente de forma independiente y una vez integrados. Además, se debe comprobar que la aplicación software cumple con las expectativas del cliente y que se integra sin problemas en el entorno productivo del mismo, si lo hubiere.

En caso de que así fuese, se procede al despliegue y explotación del producto. La fase de *despliegue* [Bays, 1999, Nygard, 2007, Humble and Farley, 2010], aunque en principio pueda parecer trivial, suele conllevar una gran cantidad de trabajo, pues hay que crear los soportes tanto lógicos como físicos para distribuir el producto, integrar el producto en el sistema productivo real del cliente, y si ello fuese necesario, crear manuales para los usuarios o impartir cursos de formación, entre otras actividades.

Una vez entregado el producto, se procede a la fase de *mantenimiento* [Grubb and Takang, 2003, April and Abran, 2008, Piattini et al., 2000] del mismo. Esta necesidad de mantenimiento puede deberse a: (1) la necesidad de subsanar errores que pudiesen aparecer durante la operación del producto, (2) la adecuación y adaptación del producto a posibles cambios que se puedan producir en el entorno, como por ejemplo, el cambio de la peseta al euro, o (3) la incorporación de posibles mejoras.

Junto a estas actividades se llevan a cabo otra serie de actividades transversales a todas ellas, que son la *gestión de la configuración*, la *planificación y gestión del proyecto de ingeniería* y la *gestión y el control de la calidad*.

Durante el desarrollo del proyecto se producirán diversos artefactos, tales como diferentes documentos de diseño y módulos software. Dichos artefactos se irán modificando conforme se vayan subsanando errores, incorporando mejoras o integrándolos con otros artefactos, por lo que aparecerán diversas versiones por cada uno de ellos. Como consecuencia, se creará una compleja red de dependencias entre cada versión específica de un artefacto y versiones específicas de otros artefactos, pues no toda versión de un artefacto será compatible con todas las versiones de los otros artefactos. Además, se deben diseñar mecanismos para encontrar y recuperar versiones concretas de un conjunto de artefactos. La adecuada gestión de esta compleja red de

dependencias es el objetivo principal de la *gestión de la configuración* [Jonassen Hass, 2003, Leon, 2004, Pilato et al., 2008].

La *planificación y gestión del proyecto de ingeniería* [Project Management Institute, 2004, Highsmith, 2009, Kerzner, 2009, Schwalbe, 2010] se ocupa de los aspectos generales de la gestión del proyecto software, tales como el control de costes [Boehm et al., 2000], la identificación y gestión de los riesgos [Kendrick, 2009], la gestión de los recursos humanos [DeMarco and Lister, 1999] o la supervisión de la evolución real del proyecto [Fleming and Koffleman, 2010]. Las técnicas que se aplican en estos casos no difieren en muchas ocasiones de las técnicas aplicadas en otras ingenierías. Por ejemplo, para la gestión de riesgos se usan matrices de análisis DAFO (*Debilidades, Amenazas, Fortalezas y Oportunidades*) [Fine, 2009], al igual que en Ingeniería Civil o Mecánica.

La *gestión y el control de la calidad* [Kan, 2002, Duvall et al., 2007, García et al., 2008, Jones, 2008, del Peso et al., 2008, Calero et al., 2010, Piattini et al., 2011, Chrissis et al., 2011] se encarga de la definición de los procesos y técnicas que permiten a un equipo de desarrollo medir y evaluar ciertos parámetros que son de interés para una empresa. Dichas técnicas pueden aplicar tanto al producto desarrollado como al proceso seguido para su desarrollo. En relación al producto, el objetivo es verificar y cuantificar que el producto satisface una serie de restricciones de calidad, como, por ejemplo, un cierto grado de usabilidad [Fernández et al., 2011] . En relación al proceso, el objetivo es definir y analizar procesos de desarrollo software bien definidos. Si no estuvieran bien definidos, dichos procesos no serían repetibles y difícilmente analizables. A partir del análisis de las medidas obtenidas de un proceso de desarrollo software es posible su rendimiento y productividad. Este análisis debe permitir identificar los puntos débiles de dicho proceso, los cuales deberán mejorarse conforme a a un proceso de mejora continua.

Todas estas fases pueden combinarse de diversas maneras, de acuerdo a diferentes *metodologías de desarrollo software*. Estas metodologías especifican cómo combinar dichas fases en aras de aumentar la productividad de los procesos empleados y la calidad de los productos construidos. Existe actualmente una amplia variedad de metodologías de desarrollo de productos software, tales como el Proceso Unificado [Jacobson et al., 1999] o la familia de metodologías ágiles [Cockburn, 2006], entre las que se incluyen Scrum [Pichler, 2010] o el Desarrollo Dirigido por Pruebas (*TDD, Test Driven Development*) [Beck, 2002].

Dado el auge en los últimos años de las metodologías iterativas, en la Figura 1 se han dispuesto deliberadamente las distintas fases que conforman un proyecto software en forma circular. Siguiendo un enfoque iterativo, un producto software se construye mediante diversas ciclos o iteraciones, en cada una de las cuales se llevan a cabo las diferentes fases expuestas con anterioridad, desde requisitos hasta el despliegue. En cada fase se añaden nuevas

funcionalidades al producto construido. Las acciones de mantenimiento se suelen contemplar como iteraciones especiales dentro de este proceso de desarrollo, las cuales se realizan una vez desplegado el producto y cuyo objetivo es subsanar errores, adaptar el sistema a cambios en el entorno o introducir mejoras.

Dependiendo de cada metodología, ciertas etapas podrían aparecer de forma ligeramente modificada, al igual que también podrían añadirse o eliminarse etapas. Por ejemplo, en el *Desarrollo de Software Basado en Componentes* [Szyperski, 2011], la fase de diseño podría omitirse, dado que normalmente se reutilizan componentes prefabricados, pero aparecería una nueva fase que sería la de selección y adaptación de dichos componentes prefabricados (en inglés *COTS (Components Off-The-Shelf)*).

Referencias

Computing Curricula 2005: The Overview Report, September 2005.

Hassan A. Afyouni. *Database Security and Auditing: Protecting Data Integrity and Accessibility* . Course Technology, April 2005.

Markus Aleksy, Axel Korthaus, and Martin Schader. *Implementing Distributed Systems with Java and CORBA*. Springer, November 2010.

Alain April and Alain Abran. *Software Maintenance Management: Evaluation and Continuous Improvement* . Wiley, April 2008.

Felix Bachmann, Len Bass, Paul Clements, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, 2 edition, October 2010.

Paul Baker, Zhen Ru Dai, Jens Grabowski, Ina Schieferdecker, and Clay Williams. *Model-Driven Testing: Using the UML Testing Profile*. Springer, 4 edition, November 2007.

Henry Balen. *Distributed Object Architectures with CORBA*. Cambridge University Press, February 2000.

Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, April 2003.

Christian Bauer and Gavin King. *Java Persistence with Hibernate*. Manning, November 2006.

Michael E. Bays. *Software Release Methodology*. Prentice Hall, July 1999.

Kent Beck. *Test Driven Development: By Example*. Addison-Wesley Professional, November 2002.

Barry W. Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Bert Steece. *Software Cost Estimation with Cocomo II*. Prentice Hall, August 2000.

Jan Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley Professional, May 2000.

Pierre Bourque and Robert Dupuis, editors. *Guide to the Software Engineering Body of Knowledge*. IEEE (Institute of Electrical and Electronics Engineers), 2004.

Don Brown, Chad Michael Davis, and Scott Stanlick. *Struts 2 in Action*. Manning, May 2008.

Ilene Burnstein. *Practical Software Testing*. Springer, June 2003.

Coral Calero, M. Angeles Moraga, and Mario Piattini. *Calidad del Producto y Proceso Software*. Ra-Ma, 2010.

Mary Beth Chrissis, Mike Konrad, and Sandra Shrum. *CMMI for Development: Guidelines for Process Integration and Product Improvement*. Addison Wesley, 3 edition, March 2011.

Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley, October 2000.

Alistair Cockburn. *Agile Software Development: The Cooperative Game*. Addison-Wesley Professional, 2006.

Krysztof Czarnecki and Ulrich Eisenecker. *Generative Programming: Methods, Tools and Applications*. Addison-Wesley Professional, June 2000.

Emilio del Peso, Mar del Peso, and Mario Piattini. *Auditoría de Tecnologías y Sistemas de Información*. Ra-Ma, 2008.

Tom DeMarco and Timothy Lister. *Peopleware: Productive Projects and Teams*. Dorset House, 2 edition, February 1999.

Paul M. Duvall, Steve Matyas, and Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley, July 2007.

Bryon K. Ehlmann. *Object Relationship Notation (ORN) for Database Applications: Enhancing the Modeling and Implementation of Associations*. Springer, June 2009.

James Elliott, Timothy M. O'Brien, and Ryan Fowler. *Harnessing Hibernate*. O'Reilly, April 2008.

Ramez Elsmari and Sham Navathe. *Fundamentals of Database Systems*. Addison Wesley, June 2010.

Adrián Fernández, Silvia Abrahão, and Emilio Insfrán. A Web Usability Evaluation Process for Model-Driven Web Development. In Haralambos Mouratidis and Colette Rolland, editors, *Proc. of the 23rd Int. Conference on Advanced Information Systems Engineering (CAiSE)*, volume 6741 of *Lecture Notes in Computer Science*, pages 108–122, London (United Kingdom), June 2011. doi: http://dx.doi.org/10.1007/978-3-642-21640-4_10.

Lawrence G Fine. *The SWOT Analysis: Using your Strength to Overcome Weaknesses Using Opportunities to overcome Threats*. CreateSpace, October 2009.

Quentin W. Fleming and Joel M. Koffleman. *Earned Value Project Management*. Project Management Institute, 2010.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, November 1994.

Félix O. García, Javier Garzás, Marcela F. Genero, and Mario Piattini. *Medición y Estimación del Software: Técnicas y Métodos para Mejorar la Calidad y la Productividad*. Ra-Ma, 2008.

Michael Gertz and Sushil Jajodia. *Handbook of Database Security: Applications and Trends*. Springer, November 2010.

Penny Grubb and Armstrong A. Takang. *Software Maintenance: Concepts and Practice*. World Scientific, July 2003.

Jim Highsmith. *Agile Project Management: Creating Innovative Products*. Addison-Wesley Professional, 2 edition, July 2009.

Elizabeth Hull, Ken Jackson, and Jeremy Dick. *Requirements Engineering*. Springer, 3 edition, 2010 2011.

Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, August 2010.

Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley Professional, February 1999.

Anne Mette Jonassen Hass. *Configuration Management Principles and Practice*. Addison-Wesley Professional, January 2003.

Capers Jones. *Estimating Software Costs: Bringing Realism to Estimating*. McGraw-Hill, April 2007.

Capers Jones. *Applied Software Measurement: Global Analysis of Productivity and Quality*. McGraw-Hill, 3 edition, April 2008.

Stephen H. Kan. *Metrics and Models in Software Quality Engineering*. Addison Wesley, 2 edition, September 2002.

Tom Kendrick. *Identifying and Managing Project Risk: Essential Tools for Failure-Proofing Your Project*. American Management Association, 2 edition, February 2009.

Harold Kerzner. *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*. Wiley, 10 edition, March 2009.

Alexis Leon. *Software Configuration Management Handbook*. Artech Print, 2 edition, December 2004.

Glenford J. Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing*. Wiley, November 2011.

Peter Naur and Brian Randell, editors. *Software Engineering: Report of a conference sponsored by the NATO Science Committee*, Garmisch (Germany), October 1968.

Michael T. Nygard. *Release It!: Design and Deploy Production-Ready Software*. Pragmatic Bookshelf, April 2007.

Mario Piattini, José Villalba, Francisco Ruiz, Teresa Bastanchury, Macario Polo, Miguel Angel Martínez, and César Nistal. *Mantenimiento del Software. Modelos, Técnicas y Métodos para la Gestión del Cambio*. Ra-Ma, November 2000.

Mario Piattini, José Calvo-Manzano, and Joaquín Fernández. *Análisis y Diseño de Aplicaciones Informáticas de Gestión*. Ra-Ma, 2003.

Mario Piattini, Esperanza Marcos, Coral Calero, and Belen Vela. *Tecnología y Diseño de Bases de Datos*. Ra-Ma, 2006.

Mario Piattini, Felix O. Garcí, and Ignacio Guzmán. *Calidad de Sistemas de Información*. Ra-Ma, 2 edition, 2011.

Roman Pichler. *Agile Product Management with Scrum: Creating Products that Customers Love*. Addison-Wesley Professional, 2010.

Josef Pieprzyk, Thomas Hardjono, and Jennifer Seberry. *Fundamentals of Computer Security*. Springer, December 2010.

C. Michael Pilato, Ben Collins-Sussman, and Brian W. Fitzpatrick. *Version Control with Subversion*. O'Reilly, 2 edition, September 2008.

Klaus Pohl. *Requirements Engineering: Fundamentals, Principles, and Technique*. Springer, July 2010.

Roger Pressman. *Software Engineering: A Practitioner's Approach*. Mc Graw Hill, January 2009.

Project Management Institute. A Guide to the Project Management Body of Knowledge (PMBOK Guide). Technical report, American National Standard, 2004.

Kevin Roebuck. *Object-Relational Mapping (ORM)*. Tebbo, June 2011.

Kathy Schwalbe. *Information Technology Project Management*. Course Technology, 6 edition, July 2010.

Bill Scott and Theresa Neil. *Designing Web Interfaces: Principles and Patterns for Rich Interactions*. O'Reilly Media, January 2009.

Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, April 1996.

Ian Sommerville. *Software Engineering*. Addison Wesley, 9 edition, March 2010.

Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Professional, June 2011.

Richard N. Taylor, Nenad Medvidovic, and EricM. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley, January 2009.

Jenifer Tidwell. *Designing Interfaces*. O'Reilly, January 2011.

Craig Walls. *Spring in Action*. Mann, 3 edition, June 2011.

Thomas Weigert and Paul Dietz. Automated Generation of Marshaling Code from High-Level Specifications. In Rick Reed and Jeanne Reed, editors, *Proc. of the 11th Int. SDL Forum (SDL)*, volume 2708 of *Lecture Notes in Computer Science (LNCS)*, Stuttgart (Germany), July 2003. doi: http://dx.doi.org/10.1007/3-540-45075-0_22.

Colin Willcock, Thomas Deiß, Stephan Tobies, Stefan Keil, Federico Engler, Stephan Schulz, and Anthony Wiles. *An Introduction to TTCN-3*. Wiley, 2 edition, April 2011.