DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

El proceso de desarrollo

Fernando Berzal, berzal@acm.org

El proceso de desarrollo

- Tareas y actividades
  - Modelos de proceso
  - Actividades de gestión
  - Mitos y realidades
- Gestión de las distintas etapas de un proyecto
  Planificación, análisis de requisitos, diseño, implementación, pruebas, documentación, control de calidad (revisiones técnicas), entrega/despliegue, mantenimiento
- Evaluación y mejora de procesos
- Apéndice: Modelos & SWEBOK

Tareas y actividades

"A process defines who is doing *what, when,* and *how* to reach a certain goal."

-- Ivar Jacobson, Grady Booch & James Rumbaugh

"Good ideas are not adopted automatically. They must be driven into practice with courageous patience."

-- **Hyman Rickover**, American admiral (1900–1986)

"Managing is harder than programming, because making people do what you want is far more difficult than making computers do what you want."

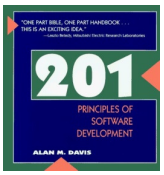
Tareas y actividades

Modelos de proceso de desarrollo de software

- Cleanroom
- PSP
- TSP
- CMMI
- UP
- RAD
- XP
- DSDM
- Scrum
- Kanban
- …

Tareas y actividades

No existe un modelo de proceso que funcione bien para todos los proyectos de una organización.



Cada proyecto debe seleccionar un modelo adecuado a sus características, teniendo en cuenta:

- La cultura corporativa.
- Su nivel de riesgo.
- Su área de aplicación.
- La volatilidad de los requisitos.
- Hasta qué punto se conocen los requisitos.

...

L. Alexander & A. Davis: "Criteria for the Selection of a Software Process Model", IEEE COMPSAC'1991

Tareas y actividades

Al adoptar (o adaptar) un proceso de desarrollo de software, el gestor del proyecto debe establecer:

- El flujo de **actividades** y el conjunto de tareas para cada actividad, así como las dependencias entre ellas.
- Los **productos** que se han de obtener como resultado de cada tarea [work products].
- La forma en que se realizarán las tareas de control de calidad asociadas al proyecto [**QA**: quality assurance].
- Los mecanismos de **comunicación** y colaboración con el cliente y otros "stakeholders" del proyecto.
- El grado de detalle y **rigor** con el que se aplicará un proceso de desarrollo de software.
- El grado de autonomía con el que trabajará el equipo.

Tareas y actividades

La prácticas de desarrollo y gestión de proyectos dependen del contexto:

- Nunca se debe decidir qué hacer sin entender antes el contexto en el que se trabajará.
- Adoptar prácticas al pie de la letra (según el libro) sólo es aconsejable si la adopción va seguida inmediatamente de un proceso de aprendizaje que. permita ajustar esas prácticas al contexto local.

Como en la canción, "depende, todo depende…"

Tareas y actividades

Como en la resolución de cualquier problema [Polya: How to Solve It]

- Comprender el problema (comunicación y análisis).
- Plantear una solución (modelado y diseño).
- Ejecutar el plan (implementación).
- Examinar el resultado (pruebas & QA)

Tareas y actividades

7 principios generales
[David Hooker, 1996]
http://c2.com/cgi/wiki?SevenPrinciplesOfSoftwareDevelopment

1. La razón por la que existe todo: Ofrecer VALOR.
2. KISS [Keep It Simple, Stupid!].
3. Visión (esencial para el éxito de un proyecto).
4. Lo que una persona produce, otra lo consume.
5. Mantenibilidad (capacidad de adaptación en el futuro).
6. Reutilización planificada (ROI)
7. ¡Piensa!

Tareas y actividades

Actividades transversales de gestión
(comunes a las distintas fases de un proyecto):

- Seguimiento y control del proyecto.
  p.ej. Métricas
- Gestión de riesgos [risk management]
- Gestión de la configuración del software [SCM]
- Control de calidad [QA: Quality Assurance]
  p.ej. Revisiones técnicas

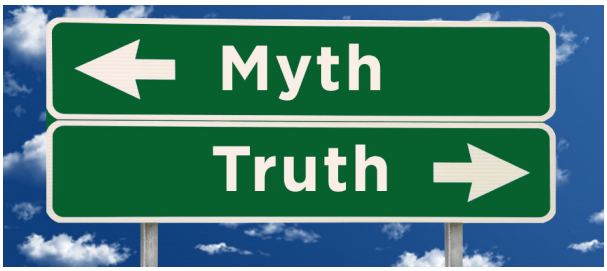
Tareas y actividades

Actividades concretas de gestión
(específicas para las distintas fases de un proyecto):

- Planificación
- Análisis de requisitos
- Diseño
- Implementación
- Despliegue
- Mantenimiento

Mitos y realidades

Afectan a gestores de proyectos, ingenieros de software, clientes y otros "stakeholders" no técnicos.



Son creíbles porque, a menudo, tienen parte de verdad…

… pero conducen inevitablemente a malas decisiones.

Mitos y realidades

MITO

Existen estándares que ya establecen los procedimientos que hay que seguir en todo momento.

REALIDAD

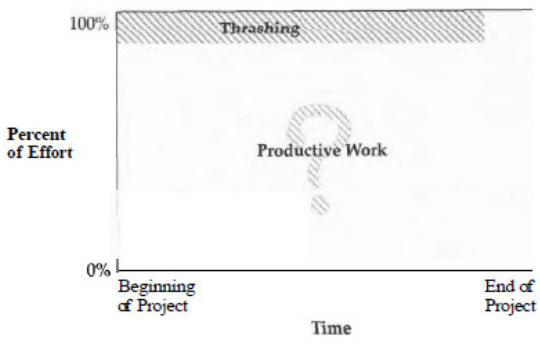
Puede que el estándar necesario exista, pero

- ¿se usa?
- ¿es consciente el equipo de su existencia?
- ¿es completo?
- ¿se puede adaptar adecuadamente?

Mitos y realidades

MITO

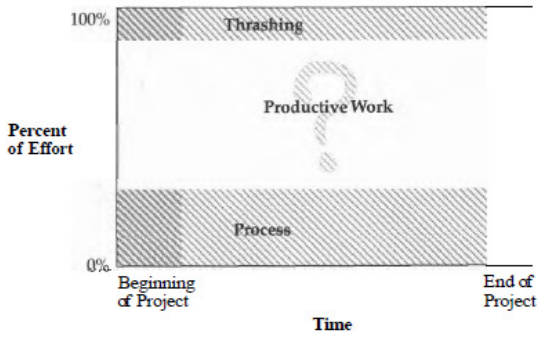
Ignorar el proceso aumenta la proporción de trabajo productivo en el proyecto.



Steve McConnell: "Software Project Survival Guide", 1997

Mitos y realidades

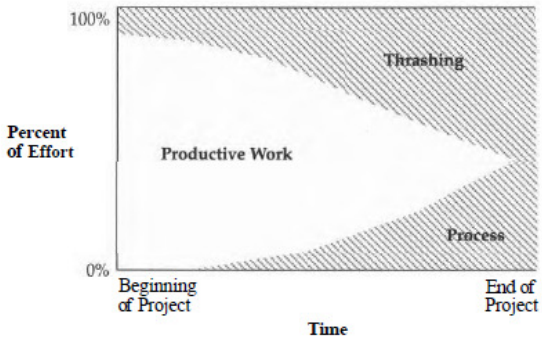
MITO
Prestar atención al proceso disminuye la proporción de trabajo productivo en el proyecto.



Steve McConnell: "Software Project Survival Guide", 1997

Mitos y realidades

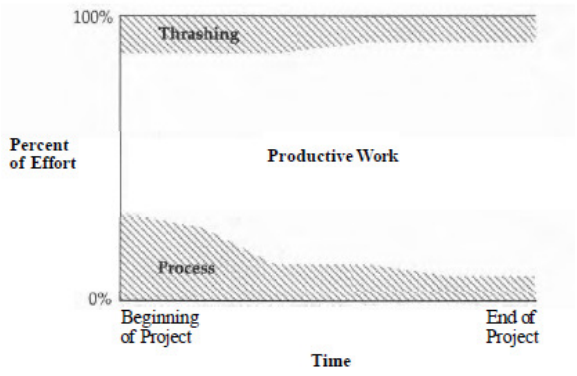
REALIDAD
Si se presta poca atención al proceso de desarrollo.



Steve McConnell: "Software Project Survival Guide", 1997

Mitos y realidades

REALIDAD
Si presta atención al proceso desde el principio
(conforme se adquiere experiencia y el proceso se afina).

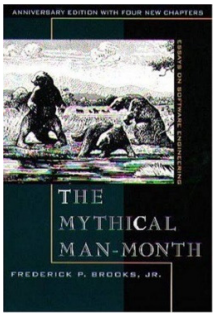


Steve McConnell: "Software Project Survival Guide", 1997

Planificación

MITO ["Mongolian horde"]
Si el proyecto se retrasa,
siempre podemos añadir nuevos programadores.

REALIDAD
Ley de Brooks
Añadir gente a un proyecto retrasado
sólo lo retrasa aún más.



Planificación

En procesos de tipo iterativo, la planificación se realiza al comienzo de cada iteración/sprint y no debe olvidar las siguiente actividades:

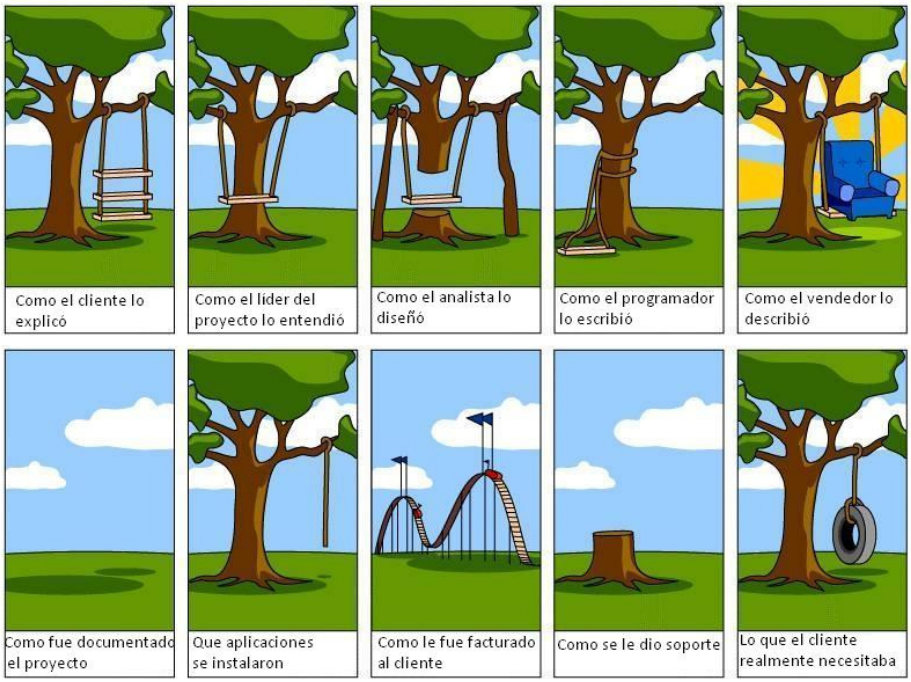
- Revisión de requisitos.
- Diseño detallado.
- Implementación.
- Revisiones de código.
- Creación de casos de prueba.
- Actualización de la documentación de usuario.
- Corrección de defectos.
- Integración.
- Actividades transversales (seguimiento del proyecto, gestión de riesgos…)

Planificación

Timeboxing / sprints

- Una forma de controlar el ritmo del proyecto es descomponerlo en períodos de tiempo de duración fija (time boxes o "sprints").
- El software se entrega periódicamente al final de cada período.
- Dentro de cada período se mantienen bajo control las solicitudes de cambios.

Análisis de requisitos



Análisis de requisitos

MITO

Una descripción general de los objetivos del proyecto es suficiente para comenzar a trabajar en el proyecto (ya se concretarán los detalles más adelante).

REALIDAD

Aunque no siempre es posible establecer un conjunto completo y estable de requisitos, una especificación ambigua garantiza el desastre. La elicitación de requisitos no ambiguos requiere una comunicación efectiva y continua entre cliente y desarrollador.

Análisis de requisitos

Prototipado

Reducción de riesgos

Si el cliente tiene una necesidad pero es incapaz de articular sus detalles, desarrollar un prototipo puede ser un primer paso…

"Plan to throw one away. You will do that, anyway."
-- **Frederick P. Brooks**: "The Mythical Man-Month", 1975

(cita original de Winston Royce: "Managing the Development of Large Software Systems," WESCON'1970, reimpreso en ICSE'1987)

Análisis de requisitos

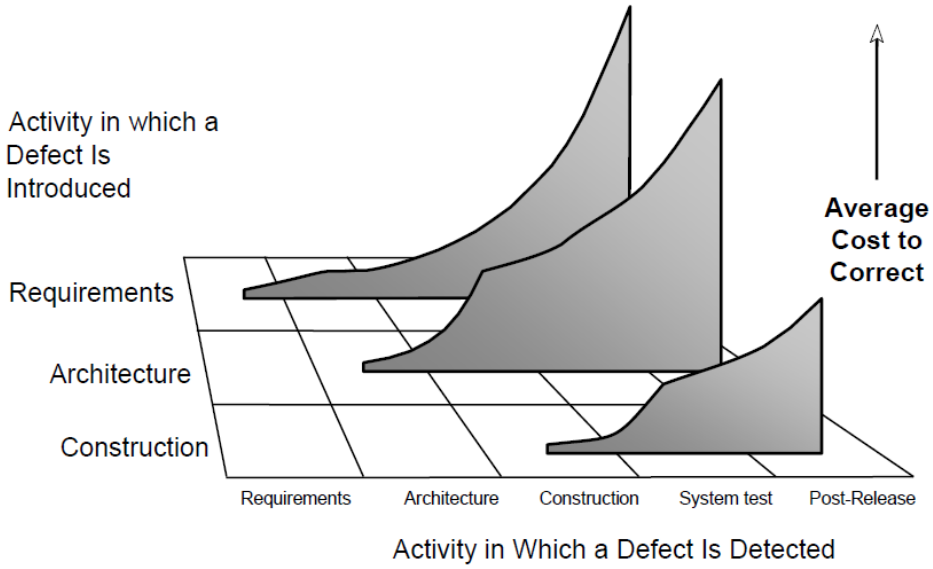
MITO

Los requisitos del software cambian continuamente, pero los cambios son fáciles de acomodar porque el software es maleable.

REALIDAD

El impacto de los cambios varía en función de la etapa del proyecto en la que se introducen.

Análisis de requisitos



Steve McConnell: "Software Project Survival Guide", 1997
Barry Boehm: "Software Engineering," IEEE TC 25(12):1226-1241, **1976**

Análisis de requisitos

MITO

El cliente sabe lo que quiere…

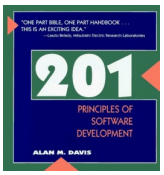
REALIDAD

El cliente no sabe lo que quiere,
pero sabe lo que no quiere en cuanto lo ve.

"… much of the essence of building a program is in fact the debugging of the specification."
— Frederick P. Brooks Jr.: "The Mythical Man-Month"

Análisis de requisitos

El cambio es inevitable



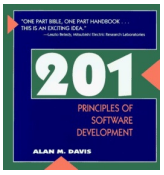
"No matter where you are in the system [development] life cycle, the system will change, an d the desire to change it will persist throughout the life cycle"

-- **Edward Bersoff, V. Henderson & S. Siegel**: "Software Configuration Management," 1980.

Análisis de requisitos

Trazabilidad

¿Por qué se incluyó un requisito?



Cuando se tome una decisión con respecto a la inclusión de un requisito, resulta aconsejable registrar su origen.

Tom Gilb: "Principles of Software Engineering Management", 1988

Diseño

"Humans have an attraction for complexity, and programmers especially so."

— Russ Daniels, VP of Engineering, HP

"You know you've achieved perfection in design, not when you have nothing more to add, but when you have nothing more to take away."

— Antoine de Saint-Exupéry

"There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies and the other is to make it so complicated that there are no obvious deficiencies"

— C.A.R. Hoare

Diseño

Diseño arquitectónico

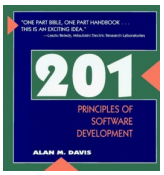
"Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be cancelled."

— Eoin Woods

"Good design adds value faster than it adds cost."

— Thomas C. Gale, Chrysler Chief of Automotive Design and Product Development

Diseño



Diseño arquitectónico

Características deseables:

- Modular
  (partes intercambiables)
- Bajo control intelectual
  (permite su comprensión completa).
- Integridad conceptual
  (número limitado de "formas" usadas uniformemente).
- Maleable
  (flexible para acomodar requisitos no anticipados)

B. Witt, F. Baker & E. Merritt:
"Software Architecture and Design", 1994

Diseño

Ley de Conway

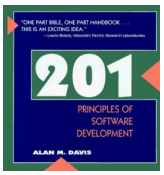
Melvin E. Conway: **"How do Committees Invent?"**,
Datamation 14(5):28–31, April 1968

Las organizaciones que diseñan sistemas… están limitadas a producir diseños que son copias de las estructuras de comunicación de esas organizaciones.

"If you have four groups working on a compiler, you'll get a 4-pass compiler"
-- Eric S. Raymond: The New Hacker's Dictionary

Diseño

Distancia intelectual



- Edsger Dijsktra definió la distancia intelectual como la distancia entre el problema del mundo real y la solución computerizada a ese problema.
- Cuanto menor sea la distancia intelectual, más sencillo será mantener el software, de ahí la popularidad de la orientación a objetos o de MDSD ["model-driven software development"].

Richard Fairley: "Software Engineering Concepts", 1985

Implementación

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."

-- Martin Fowler

Eagleson's Law: "Any code of your own that you haven't looked at for six or more months might as well have been written by someone else."

Weinberg's Second Law: "If engineers built buildings the way programmers wrote programs, the first woodpecker to come along would destroy civilization."

Implementación

MITO

Si se deciden externalizar [outsource] tareas de desarrollo, podemos olvidarnos y dejar que la empresa externa se encargue de todo.

REALIDAD

Si una organización no comprende cómo gestionar internamente proyectos de desarrollo de software, invariablemente lo pasará mal cuando los externalice.

Implementación

MITO

Una vez que implementamos un programa y conseguimos que funcione, nuestro trabajo ha terminado.

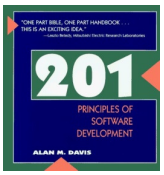
REALIDAD

Los datos indican que del 60% al 80% del esfuerzo dedicado al desarrollo de software se realiza después de que haya sido entregado al cliente por primera vez.

Implementación

Reutilización

Si es posible, reutilice en vez de construir



La técnica más efectiva para reducir costes y riesgos en el desarrollo de software es reutilizar software en vez de construirlo desde cero.

Frederick P. Brooks: "No Silver Bullet: Essence and Accidents of Software Engineering", IEEE Computer 20(4):10-19, April 1987

Implementación

Estándares de codificación

- Layout del código
- Comentarios
- Identificadores (variables, funciones, ficheros…)
- Complejidad (longitud, estructuras de control…)
- Herramientas y bibliotecas (versiones concretas)
- Organización de los ficheros del proyecto
- Formas de indicar fragmentos incompletos (p.ej. TBD).

Deben mantenerse más o menos iguales de proyecto a proyecto dentro de una organización y forzarse su uso mediante revisiones de código [code reviews].

Implementación

Refactorización

Mejorar el código sin cambiar su funcionalidad

- Una forma de reducir la "deuda técnica" [technical debt] que se va acumulando conforme el sistema crece y se hace más complejo.
- La reducción de la "deuda técnica" es una forma de interés compuesto (la fuerza más poderosa del Universo según Einstein): la realización continua de pequeñas refactorizaciones incrementales ayuda a mantener la calidad del sistema y ahorra costes de mantenimiento futuros.

Implementación

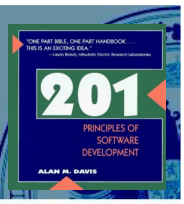
Integración: Procedimiento recomendado

TABLE 14-1 RECOMMENDED INTEGRATION PROCEDURE

1. Developer develops a piece of code.
2. Developer unit tests the code.
3. Developer steps through every line of code, including all exception and error cases, in an interactive debugger.
4. Developer integrates this preliminary code with a private version of the main build.
5. Developer submits code for technical review.
6. Developer informally turns code over to testing for test case preparation.
7. Code is reviewed.
8. Developer fixes any problems identified during the review.
9. Fixes are reviewed.
10. Developer integrates final code with the main build.
11. Code is declared "complete."

Steve McConnell: "Software Project Survival Guide", 1997

Implementación



Una de las formas más efectivas de reducir el riesgo en la construcción de software es desarrollarlo incrementalmente.

Ventajas

- Menor riesgo en cada iteración.
- Ver el producto funcionando ayuda a los usuarios a la hora de determinar qué funciones les gustarían.

Limitación

- Si no se elige inicialmente la arquitectura apropiada, un rediseño completo puede ser necesario para acomodar los cambios necesarios (p.ej. prototipado).

Harlan D. Mills: "Top-Down Programming in Large Systems", in Debugging Techniques in Large Systems, **1971**

Implementación

Daily Build & Smoke Test

TABLE 14-2 DAILY BUILD AND SMOKE TEST PROCEDURE*

1. *Merge code changes.* The developer compares his or her private copy of the source files with the master source files, checking for conflicts and inconsistencies between recent changes made by other developers and the new or revised code to be added. The developer then merges his or her code changes with the master source files. The merging of code is usually supported by automated source code control tools, which warn the developer of any inconsistencies.
2. *Build and test a private release.* The developer builds and tests a private release to ensure that the newly implemented feature still works as expected.
3. *Execute the smoke test.* The developer runs the current smoke test against the private build to be sure the new code won't break the build.
4. *Check in.* The developer checks his or her private copies of the source code into the master source files. Some projects establish times during which new code can and can't be added to the daily build; for example, new code must be added no later than 5:00 p.m. and no earlier than 7:00 a.m.
5. *Generate the daily build.* The build team (or build person) generates a complete build of the software from the master sources.
6. *Run the smoke test.* The build team runs the smoke test to evaluate whether the build is stable enough to be tested.
7. *Fix any problems immediately!* If the build team discovers any errors that prevent the build from being tested (in other words, that break the build), it notifies the developer who checked the code that broke the build in, and that developer fixes the problem immediately. Fixing the build is the project's top priority.

* The process in this table starts at Step 10 in Table 14-1's Recommended Integration Procedure, which is the point where the developer is ready to check in code. It assumes that the developer has checked out source code files that need to be changed and has possibly created new files.

Pruebas

"Code without tests is bad code. It doesn't matter how well written it is; it doesn't matter how pretty or object-oriented or well-encapsulated it is. With tests, we can change the behavior of our code quickly and verifiably. Without them, we really don't know if our code is getting better or worse."

-- Michael Feathers,

Working Effectively with Legacy Code

Pruebas

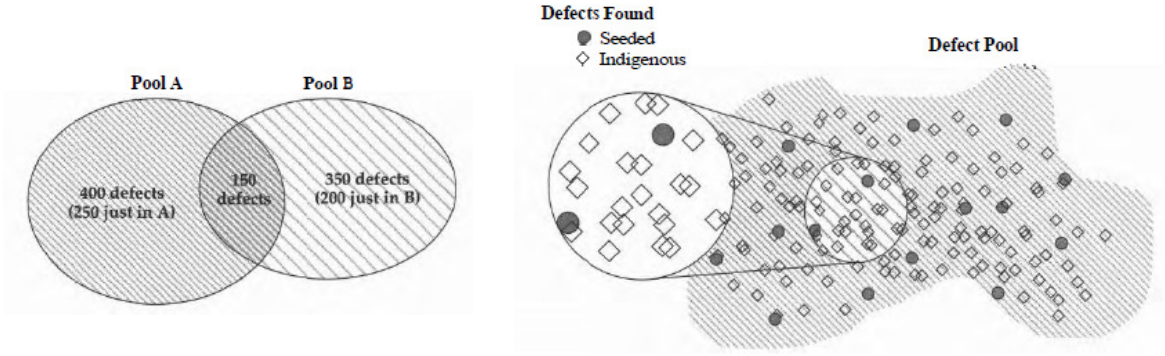
"Testing is not a phase."

— Elisabeth Hendrickson

- En paralelo con la implementación del sistema: pruebas de unidad, smoke tests...
- Un test con éxito es el que **encuentra** un error.
  J. Goodenough & S. Gerhart: "Toward a Theory of Test Data Selection", IEEE TSE 1(2):156-173, June 1975
- Identificación de módulos propensos a errores, que requieren la realización de una revisión técnica (y, posiblemente, su rediseño y reimplementación).
  A. Endres: "An Analysis of Errors and Their Causes in Systems Programs", IEEE TSE 1(2):140-149, June 1975

Pruebas

- Uso de técnicas estadísticas para decidir si el sistema está listo para su despliegue (p.ej. defect pooling, defect seeding, defect modeling…)

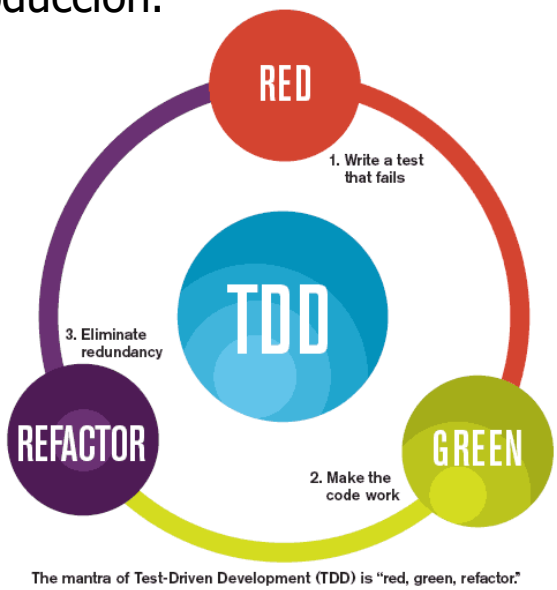


Steve McConnell: "Software Project Survival Guide", 1997

Pruebas

TDD [Test-Driven Development]

Escribir el código de los casos de prueba antes del código de producción.



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Pruebas

TDD [Test-Driven Development]

Existen pruebas de que el uso de TDD requiere un 15% más de tiempo que no utilizar TDD.

Boby George & Laurie A. Williams: **An Initial Investigation of Test Driven Development in Industry**. SAC 2003

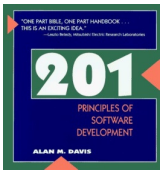
Boby George, Laurie A. Williams: **A structured experiment of test-driven development.** Information & Software Technology 46(5):337-342, 2004

También hay evidencias de que TDD reduce el número de defecto: dos estudios encontraron el número de bugs se redujo en un 24% y en un 38% con el uso de TDD.

Julio Cesar Sanchez, Laurie A. Williams & E. Michael Maximilien: **On the Sustained Use of a Test-Driven Development Practice at IBM**. AGILE 2007

Pruebas

Cobertura



Existen herramientas que permiten determinar la exhaustividad con la que el código se ha ejecutado durante las pruebas (%sentencias, %ramas, %caminos).

Un 100% de cobertura **no** garantiza que el programa sea correcto y esté libre de errores.

M. Weiser, J. Gannon & P. McMullin: "Comparison of Structural Test Coverage Metrics", IEEE Software 2(2):80-85, March 1985

Pruebas

Esfuerzo

"The only reasonable schedule estimate for testing is based not on people but on defect density."
— Jim Highsmith, BayALN, 2010

La **falacia de la ausencia de errores** (Jerry Weinberg): La presencia de muchos errores garantiza la falta de calidad del software; cero errores no da información acerca del valor del software.

Gerald M. Weinberg: "Quality Software Management, Volume 1: Systems Thinking", 1992

Pruebas

QA es fundamentalmente distinto a desarrollo:

Pensar cómo conseguir romper algo.
Vs.
Pensar cómo conseguir que funcione.

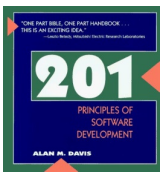
Por eso es recomendable que las personas encargadas de realizar pruebas no sean las mismas que las encargadas de su desarrollo.

G. Myers: "The Art of Software Testing", 1979

Pruebas

Do lots of user testing: Users are the best testers. (Developers are the worst testers!)

— Ron Mak, Middleware Architect of the NASA Mars Rover Mission's Collaborative Information Portal



Distintos tipos de pruebas (p.ej. caja negra y caja blanca) encuentran distintos tipos de errores:
Son complementarios
y deben utilizarse conjuntamente.

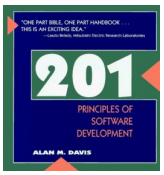
R. Dunn: "Software Defect Removal", 1984

Pruebas

"Testing by itself does not improve software quality. Test results are an indicator of quality, but in and of themselves, they don't improve it. Trying to improve software quality by increasing the amount of testing is like trying to lose weight by weighing yourself more often. What you eat before you step onto the scale determines how much you will weigh, and the software development techniques you use determine how many errors testing will find. If you want to lose weight, don't buy a new scale; change your diet. If you want to improve your software, don't test more; develop better."

— **Steve McConnell**: "Code Complete", 1993

Pruebas



Causas de los errores

Es más efectivo prevenir la aparición de errores que tener que corregirlos.

Una forma de prevenirlos es analizar sus causas cuando se detectan y tomar medidas que eviten que se repitan los mismos errores en el futuro.

Errar es humano y analizar los errores permite mejorar.

J. Kajihara, G. Amamiya & T. Saya:
"Learning from Bugs", IEEE Software 10(5):46-54, 1993

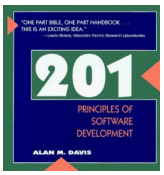
Documentación

"… the designer of a new system must not only be the implementer and the first large-scale user; the designer should also write the first user manual. . . . If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or perceived why they were important."

— Donald Knuth: "The Art of Computer Programming"

Documentación

Manuales de usuario



Una forma de medir la calidad del software es mirar el número de páginas del manual de usuario. Cuantas menos tenga, mejor es el software.

C.A.R. Hoare: "Programming: Sorcery or Science?", IEEE Software 1(2):14-15, April 1984

Control de calidad

Efecto boomerang

Círculo vicioso

- Algunas partes de QA se saltan por falta de tiempo.
- Aumenta el número de problemas de calidad.
- Aumenta el número de emergencias e interrupciones.
- Aumenta la presión sobre el equipo de desarrollo.

Gerald M. Weinberg:
"Quality Software Management," 1992.

Control de calidad

Realimentación positiva

"quality in a product pays back for itself when it leads to cost reduction and improved productivity, which in turn enables a further increase of quality"

-- Tom DeMarco & Timothy Lister:
Peopleware, Second Edition, 1999.

Revisiones técnicas

Las revisiones técnicas realizadas a lo largo del proyecto deben centrarse en encontrar:

- Defectos de diseño (corrección, completitud y comprensibilidad del diseño)
- Problemas en los requisitos (requisitos no especificados, funcionalidades no requeridas, inconsistencias en los requisitos…).
- Formas en las que se pueden satisfacer mejor los objetivos del proyecto.

Barry Boehm: "Verifying and Validating Software Requirements and Design Specifications", IEEE Software 1(1):75-88, January 1984

Revisiones técnicas

Las revisiones técnicas acercan el momento de localización de un problema al punto en el que se introdujo, lo que se traduce en un ahorro de costes.

Las revisiones técnicas (especialmente las conocidas como "inspecciones") producen un ahorro temporal del 10% al 30%:

- Cada hora empleada en la realización de inspecciones ahorra 33 horas de mantenimiento.
- Las inspecciones son hasta 20 veces más eficientes que las pruebas.

Steve McConnell: "Software Project Survival Guide", 1997

Revisiones técnicas

Realimentación

"… in a runaway project technical reviews can introduce opposing feedback loops that help getting things back under control."

-- Gerald M. Weinberg:
Quality Software Management, 1992

"Code review is as much about improving the code as it is improving the developer."

-- Roger S. Pressman:
Software Engineering: A Practitioner's Approach

Revisiones técnicas

Pair programming (programación en pareja)

- Dos programadores comparten un mismo ordenador.
- Uno programa [controlador o driver] mientras que el otro [observer, pointer o navigator] revisa lo que se va haciendo línea a línea, además de considerar la dirección "estratégica" del trabajo, idear posibles mejoras e identificar problemas futuros.
- Se puede interpretar como una revisión técnica… continua.

Entrega / Despliegue

"When a programmer says, 'it works on my machine', grab their machine and ship it to the customer!"

-- Richard Sheridan,
Bay Area Agile Leadership Network (BayALN),
May 20, 2014

Entrega / Despliegue

Release checklist

TABLE 16-1 SAMPLE ITEMS ON THE RELEASE CHECKLIST

Activities	Person Responsible
Engineering Activities	
☐ Update version strings with final version information.	Developer
☐ Remove debugging and testing code from the software.	Developer
☐ Remove seeded defects from the software.	Developer
Quality Assurance Activities	
☐ Check that all defects on current defect list have been resolved.	Tester
☐ Smoke test/regression test final build.	Tester
☐ Install program from CD ROM on clean machine.	Tester
☐ Install program from diskettes on clean machine.	Tester
☐ Install program from Internet Web site on clean machine.	Tester
☐ Install program from CD ROM/diskettes on machine with older version of program (upgrade install).	Tester
☐ Verify that the correct Windows registry entries have been created by the install program (see attached list).	Tester
☐ Verify uninstall on clean machine.	Tester
Release Activities	
☐ Freeze final list of files to be distributed.	Release team
☐ Synchronize date/time stamp on all release files.	Release team
☐ Prepare final program disks ("gold disks").	Release team

(continued)

TABLE 16-1 SAMPLE ITEMS ON THE RELEASE CHECKLIST continued

Activities	Person Responsible
☐ Verify that all files are present on the gold disks.	Release team
☐ Virus scan all release media.	Release team
☐ Surface scan the master media for bad sectors.	Release team
☐ Create a backup of the build environment and place the development environment under change control.	Release team
Documentation Activities	
☐ Verify version of readme.txt on gold disks.	Documentation team
☐ Verify version of help files on gold disks.	Documentation team
Other Activities	
☐ Verify copyright, license, and other legal materials.	Product manager, legal advisor

Entrega / Despliegue

DevOps

It is possible to deploy new code on a site within an hour.

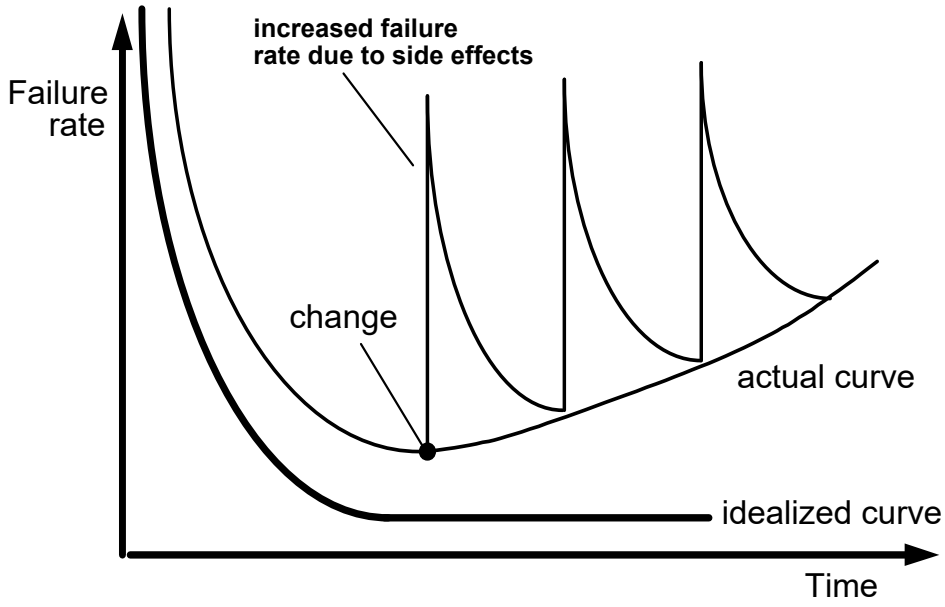
	Code	Build file	Jenkins[1] build	Develop	Test	Production
Duration	Varies	15 min	20 min	5 min	10 min	10 min
Description	Write the actual code for the service	Write the build file and define depen-dencies	Create the Jenkins[1] build and run the build job	Create new web images	Deploy the new web images in the test environment	Deploy the new web images in the production environment

[1]Jenkins is an open-source continuous-integration application that monitors execution of repeated jobs, such as building a software project.

McKinsey&Company

"Beyond agile: Reorganizing IT for faster software delivery," McKinsey & Company, September 2015
http://www.mckinsey.com/Insights/Business_Technology/Beyond_agile_reorganizing_IT_for_faster_software_delivery

Mantenimiento



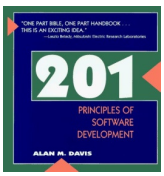
Roger S. Pressman:
"Software Engineering: A Practitioner's Approach", 7th edition, 2009

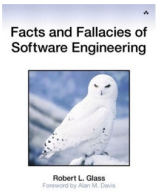
Mantenimiento

- **Mantenimiento adaptativo**
  Cambios para satisfacer necesidades cambiantes (requisitos, entorno, legislación, tecnología…).
- **Mantenimiento correctivo**
  Cambios para solucionar problemas detectados en la implementación (bugs, problemas de seguridad…).
- **Mantenimiento perfectivo**
  Cambios para resolver problemas de rendimiento (tiempo de respuesta, disponibilidad 24/7/365…).

Mantenimiento

Los sistemas software que tienen éxito requieren **más** mantenimiento que los sistemas que no tienen éxito.





- La gente utiliza su software favorito de forma innovadora y en situaciones no previstas.
- El software con éxito sobrevive sobrevive al hardware y a los procesos que se consideraron durante su creación.

B. Curtis, H. Krasner & N. Iscoe: "A Field Study of the Software Design Process for Large Systems", Communications of the ACM 31(11):1268-1287, 1988

Mantenimiento

Las 8 leyes de la evolución del software
Meir M. Lehman
https://en.wikipedia.org/wiki/Lehman%27s_laws_of_software_evolution

- Cambio continuo
  (un sistema o bien se adapta
  o bien deja de ser capaz de satisfacer a sus usuarios).
- Complejidad creciente
  (conforme un sistema evoluciona, su complejidad aumenta salvo que se trabaje para reducirla, p.ej. refactorizaciones).
- Auto-regulación
  (el proceso de evolución del sistema se autoregula)
- Conservación de la estabilidad organizativa
  (la tasa de actividad media [mantenimiento] se mantiene invariante durante su ciclo de vida)

Mantenimiento

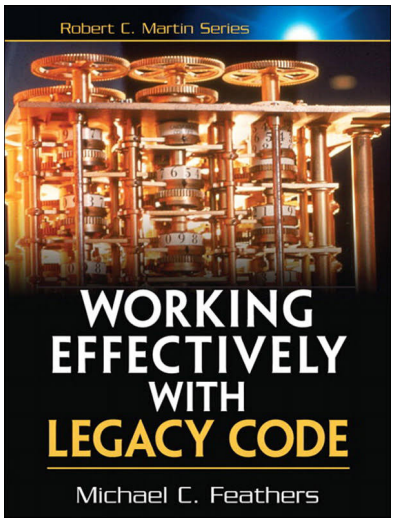
Las 8 leyes de la evolución del software
Meir M. Lehman
https://en.wikipedia.org/wiki/Lehman%27s_laws_of_software_evolution

- Conservación de familiaridad
  (todas las personas involucradas con la evolución del sistema deben mantener su maestría para una evolución satisfactoria).
- Crecimiento continuado
  (la funcionalidad del sistema debe incrementarse continuamente ppara mantener la satisfacción del usuario).
- Calidad decreciente
  (la calidad del sistema se degradará salvo que se mantenga rigurosamente y se adapte a los cambios de su entorno operativo).
- Sistema de realimentación
  (los procesos de evolución son sistemas complejos con realimentación)

Mantenimiento

El factor clave:

Testability



Michael C. Feathers:
"Working Effectively with Legacy Code", 2004.
ISBN 0131177052

Evaluación y mejora de procesos

Una práctica sencilla y útil

Checklists

El uso de listas de comprobación
puede resultar útil para verificar…
… que se ha completado realmente una tarea,
… que se ha cerrado correctamente una fase del proyecto,
… que no se han pasado por alto riesgos importantes,
… que se han considerado todos los aspectos relevantes,
…

Evaluación y mejora de procesos

Niveles de "madurez" / disciplina

- Inconsciencia [oblivious]:
  Ni siquiera sabemos que seguimos un proceso.
- Variable:
  Hacemos lo que creemos en cada momento.
- Rutina:
  Seguimos nuestras rutinas (excepto en modo pánico).
- Dirección:
  Elegimos la rutina que produce mejores resultados.
- Anticipación:
  Establecemos rutinas de acuerdo a nuestra experiencia.
- Congruencia:
  Todo el mundo se involucra en mejorar todo, todo el tiempo.

Gerald M. Weinberg: "Quality Software Management," 1992

Evaluación y mejora de procesos

- **Standard CMMI Assessment Method for Process Improvement (SCAMPI)**
  — modelo de 5 fases (iniciación, diagnóstico, establecimiento, actuación y aprendizaje).
- **CMM-Based Appraisal for Internal Process Improvement (CBA IPI)**
  — evaluación de la madurez basada en el modelo CMM.
- **Software Process Improvement and Capability dEtermination (SPICE)** ISO/IEC15504
  — Estándar que define un conjunto de requisitos para la evaluación del proceso de desarrollo de software.
- **ISO 9001:2000 for Software**
  — Estándar genérico para cualquier organización que desee mejorar la calidad de sus productos/servicios.

Evaluación y mejora de procesos

Cuidado con las exageraciones (en ambos sentidos):

"The pace of information technology (IT) change is accelerating and **agile methods** adapt to change better than disciplined methods therefore agile methods will take over the IT world."

"Software development is uncertain and the **SW-CMM** improves predictability therefore all software developers should use the SW-CMM."

Ejemplos tomados de Barry Boehm & Richard Turner:
"Balancing Agility and Discipline: A Guide for the Perplexed", 2004

Comentarios finales

Muchas metodologías y modelos no son ni más ni menos que colecciones de buenas prácticas.

- Elegidas correctamente, se refuerzan mutuamente y facilitan su implantación (formando conjuntos de "memes" interconectados, a.k.a. "memeplexes" ;-)
- Que todo el mundo haga algo, no quiere decir que sea lo más adecuado para usted. Cuando aprenda una nueva técnica, no acepte fácilmente sus exageraciones ["hype"]; sea realista en cuanto a sus recompensas y riesgos; realice experimentos antes de comprometerse (pero tampoco se permita ignorar nuevas técnicas;-)

Alan Davis: **"Software Lemmingineering"**,
IEEE Software 10(6):79-81, September 1993.

Comentarios finales

La teoría de los sistemas complejos (y un proyecto de desarrollo de software es un sistema complejo) nos dice que es imposible crear un modelo unificado de un sistema complejo…

There is no accurate (or rather, perfect) representation of the system which is simpler than the system itself. In building representations of open systems, we are forced to leave things out, and since the effects of these omissions are nonlinear, we cannot predict their magnitude.

-- Paul Cilliers: "Knowing Complex Systems", 2005

Comentarios finales

"All models are wrong, but some are useful"

-- George Box & Norman Draper.
Evolutionary Operation, 1969.

- Ningún modelo ofrece una perspectiva completa de un sistema complejo (como los proyectos de software).
- Todos los modelos, por tanto, están equivocados, aunque algunos pueden ser útiles.
- Para distintas ocasiones, es esencial en la práctica disponer de múltiples modelos complementarios y, en ocasiones, incluso contradictorios.

Comentarios finales

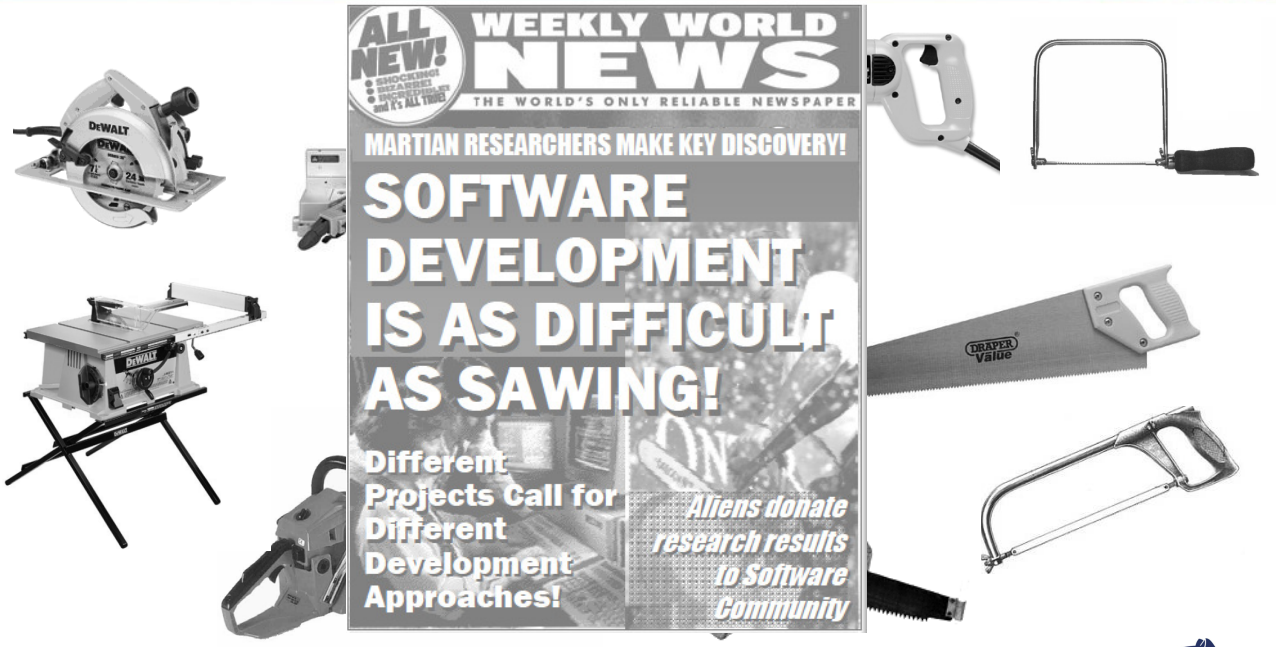
"The pure and simple truth
is rarely pure and never simple."

-- Oscar Wilde

Al final, sólo hay una verdad sencilla:

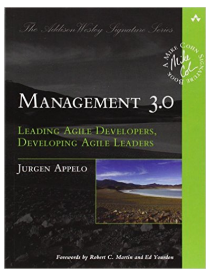
Depende

Comentarios finales



Steve McConnell (Construx Software):
"The 10 Most Important Ideas in Software Development"

Comentarios finales



PAMPHLET FOR COMPLEX PROJECTS

EACH PROBLEM HAS MULTIPLE SOLUTIONS
THERE IS NOT ONE BEST WAY TO RUN A SOFTWARE PROJECT.

SOLUTIONS DEPEND ON THE PROBLEM'S CONTEXT
THE BEST PRACTICES DEPEND ON THE PROJECT ENVIRONMENT.

CHANGING CONTEXT REQUIRES CHANGING SOLUTIONS
WHEN PROJECT ENVIRONMENTS CHANGE, PROJECTS CHANGE ACCORDINGLY.

EACH STRANGE SOLUTION IS THE BEST ONE SOMEWHERE
THERE IS ALWAYS A PLACE AND TIME FOR LESS POPULAR PRACTICES.

SOLUTIONS CHANGE THE CONTEXT AND THEMSELVES
PRACTICES CHANGE ENVIRONMENTS AND HOW WE USE PRACTICES.

SIMPLICITY NECESSITATES UNDERSTANDING COMPLEXITY
ONE MUST UNDERSTAND COMPLEXITY BEFORE APPLYING SIMPLE SOLUTIONS.

WE CANNOT PREDICT THE BEST SOLUTION
WE HAVE TO TRY PRACTICES TO KNOW IF THEY WORK IN OUR CONTEXT.

© JURGEN APPELO

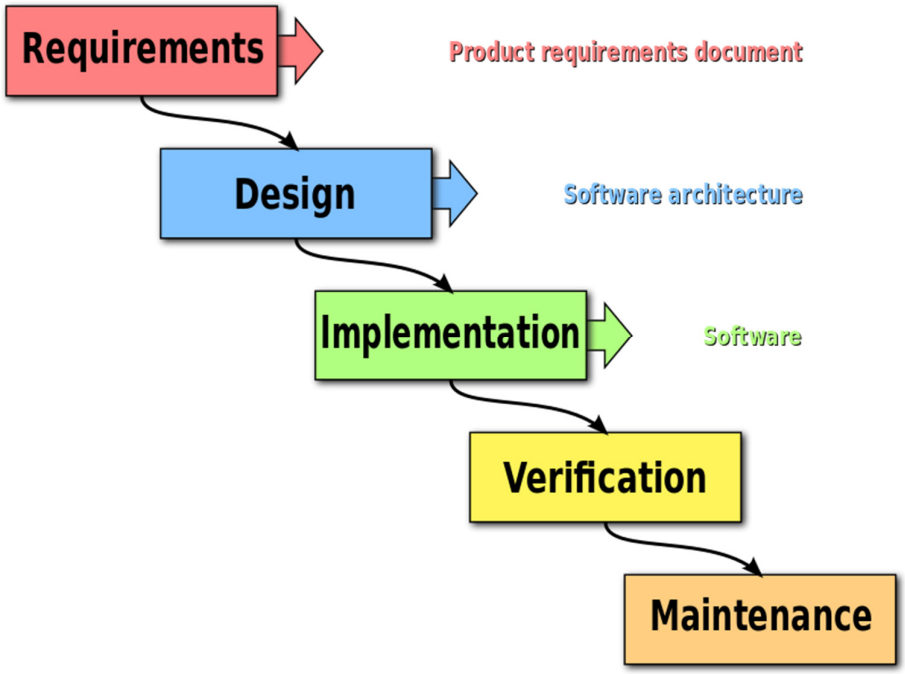
Apéndice: Modelos

"We think that software developers are missing a vital truth: most organizations don't know what they do. They think they know, but they don't know."

-- Tom DeMarco

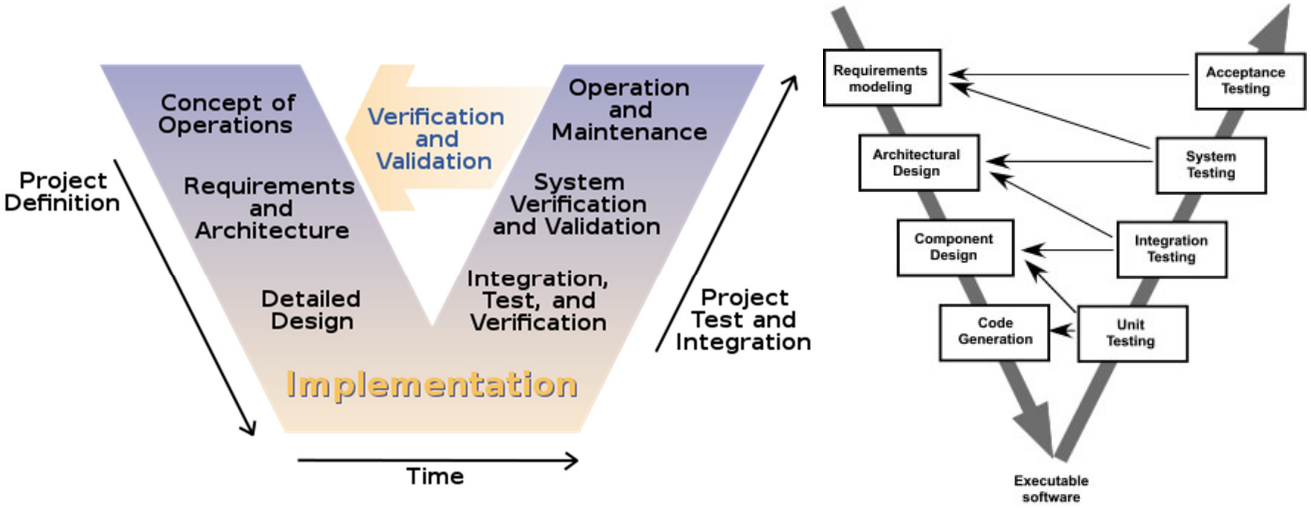
Apéndice: Modelos

Modelo en cascada [waterfall]



Apéndice: Modelos

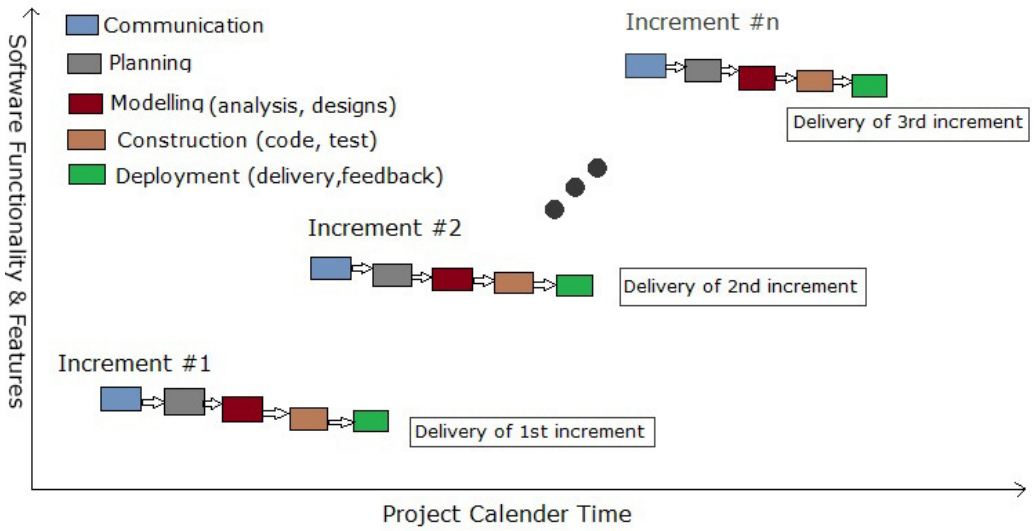
Modelo en V [V-Model]



Kevin Forsberg & Harold Mooz: "The Relationship of System Engineering to the Project Cycle," 1st Annual Symposium of National Council on System Engineering, October 1991:57–65.

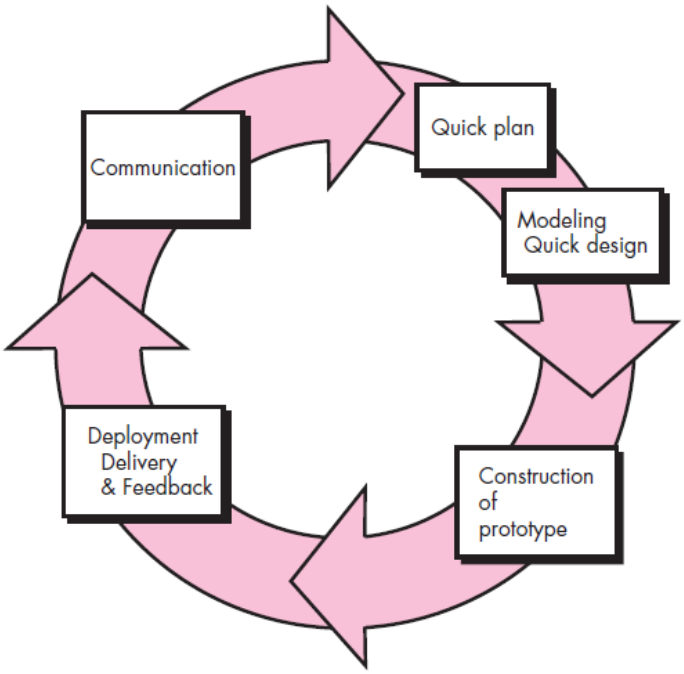
Apéndice: Modelos

Modelo incremental



Apéndice: Modelos

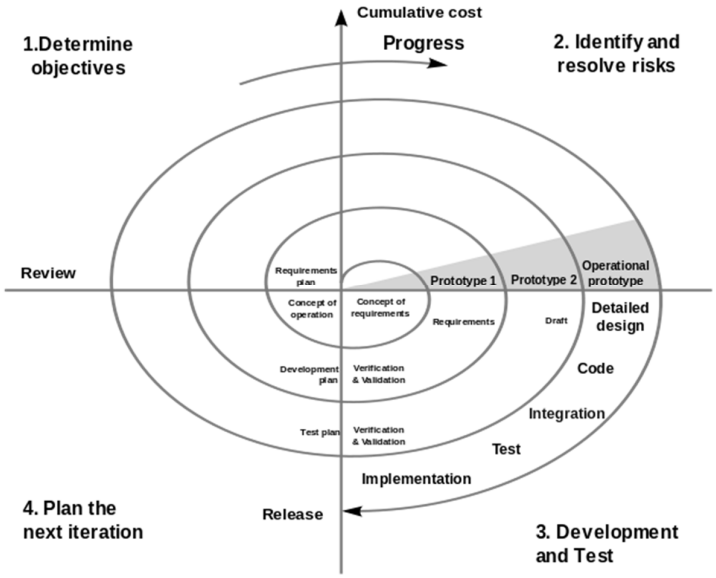
Prototipado



Roger S. Pressman:
"Software Engineering: A Practitioner's Approach", 7th edition, 2009

Apéndice: Modelos

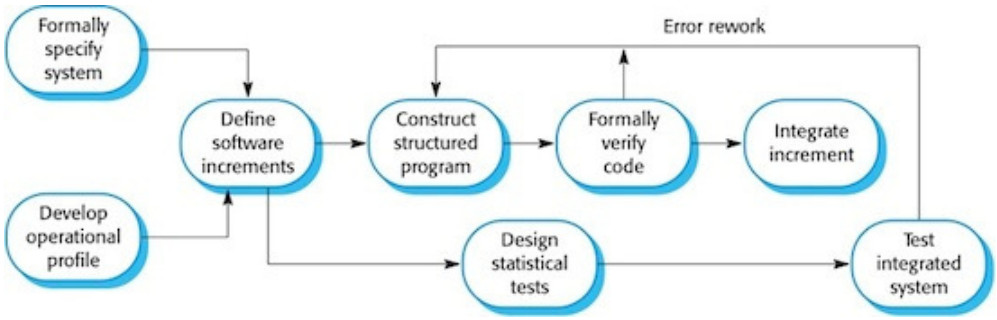
Modelo en espiral de Boehm



Barry Boehm:
"A Spiral Model of Software Development and Enhancement",
ACM SIGSOFT Software Engineering Notes 11(4):14-24, August 1986

Apéndice: Modelos

Cleanroom

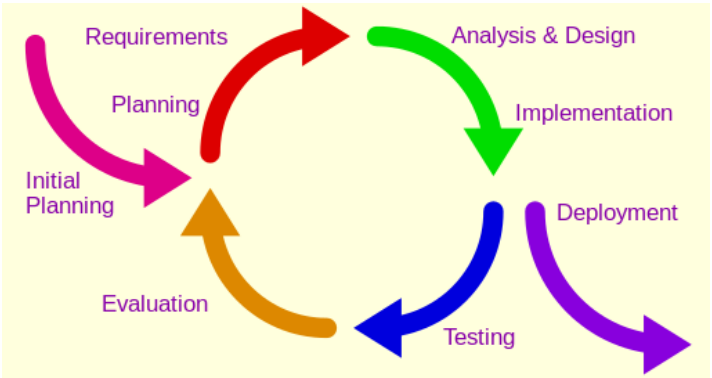


https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/Cleanroom/index.html

Harlan Mills, M. Dyer & R. Linger: "Cleanroom Software Engineering".
IEEE Software 4(5):19–25, September 1987.
DOI 10.1109/MS.1987.231413

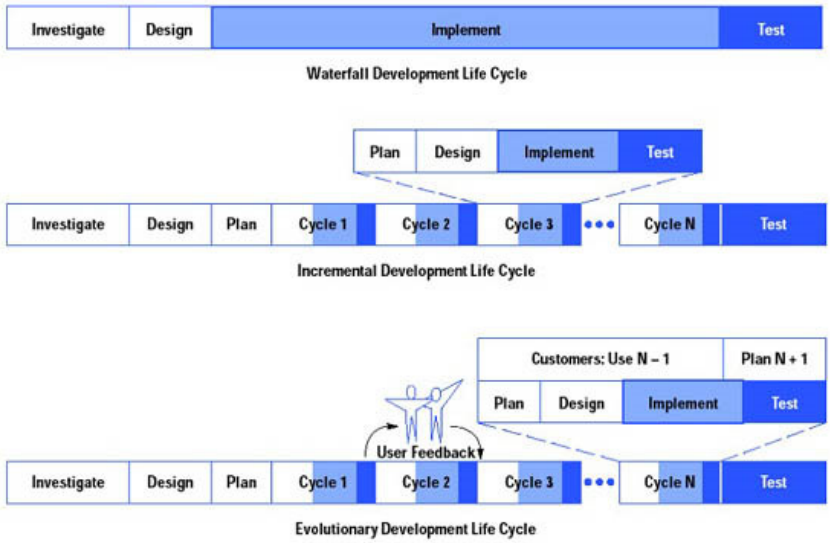
Apéndice: Modelos

IID [Iterative and Incremental Development]



Apéndice: Modelos

EVO [EVOlutionary project management]

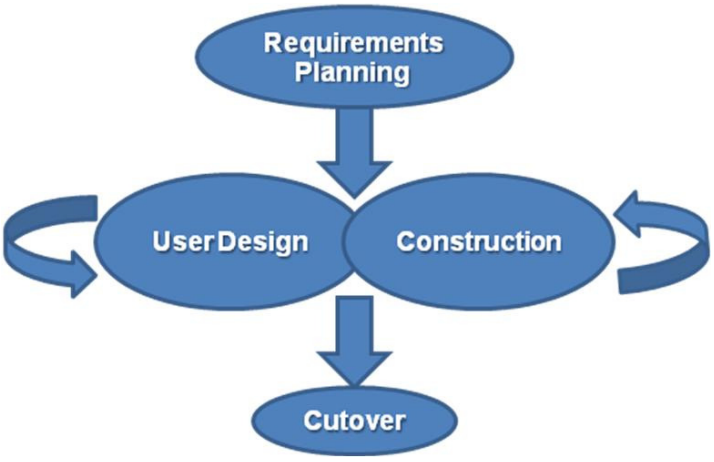


Tom Gilb:

"Principles of Software Engineering Management", 1988.
ISBN 0-201-19246-2

Apéndice: Modelos

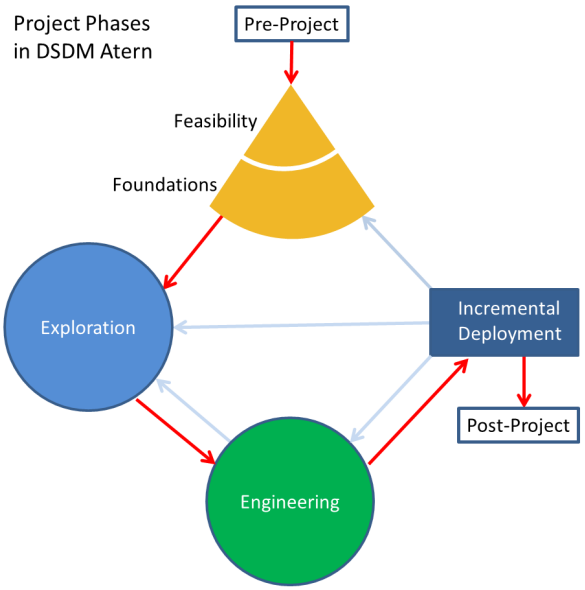
RAD [Rapid Application Development]



James Martin: "Rapid Application Development", 1991.
ISBN 0023767758.

Apéndice: Modelos

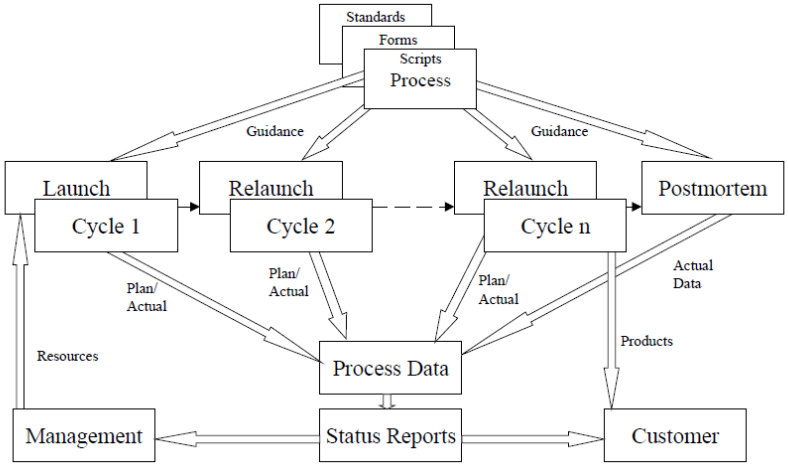
DSDM [Dynamic Systems Development Method]



DSDM Consortium, 1994

Apéndice: Modelos

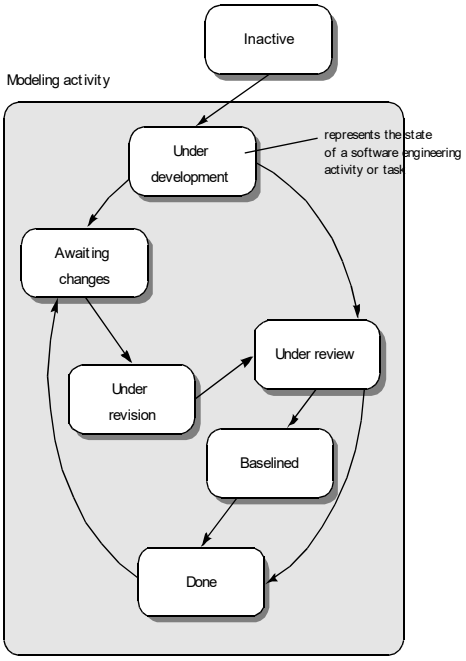
TSP [Team Software Process]



Watts S. Humphrey:
"The Team Software Process (TSP)"
Technical Report CMU/SEI-2000-TR-023, November 2000
http://www.sei.cmu.edu/reports/00tr023.pdf

Apéndice: Modelos

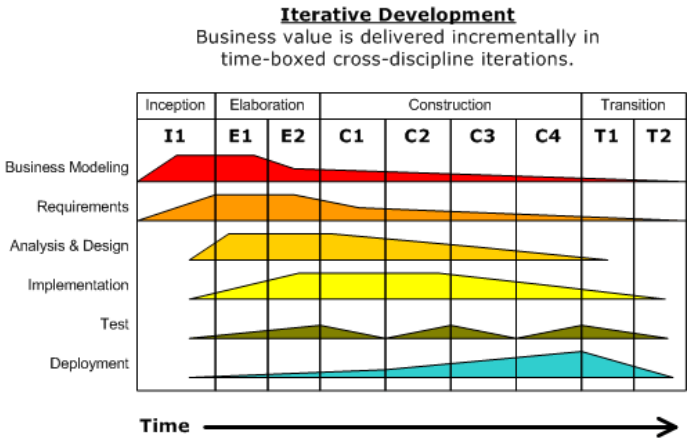
Modelo concurrente



Roger S. Pressman:
"Software Engineering: A Practitioner's Approach", 7th edition, 2009

Apéndice: Modelos

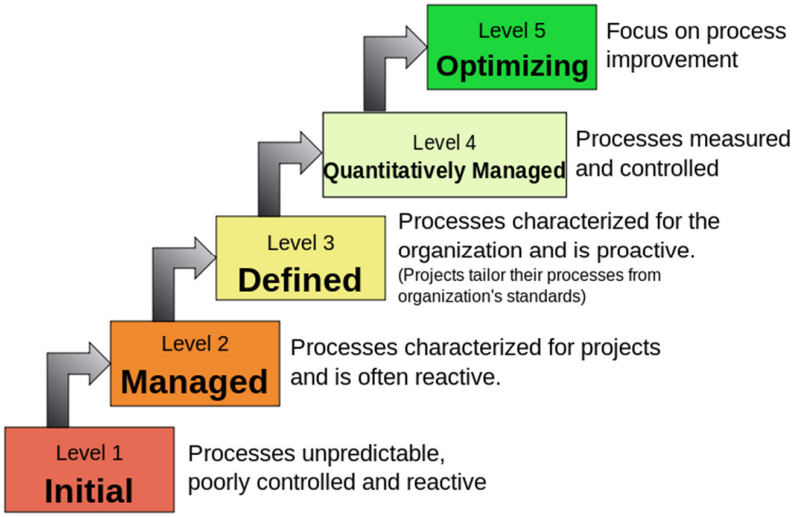
UP [Unified Process]



Ivar Jacobson, Grady Booch & James Rumbaugh:
"The Unified Software Development Process", 1999
ISBN 0-201-57169-2

Apéndice: Modelos

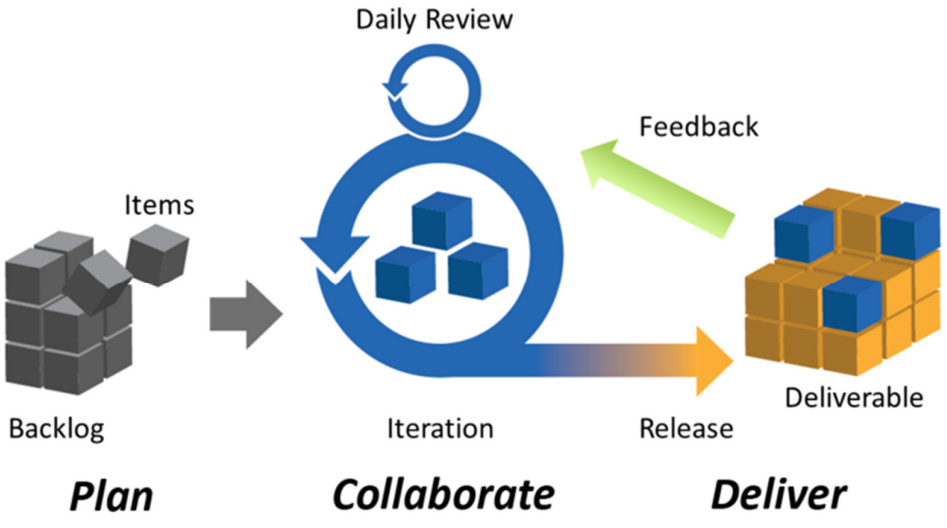
CMMI [Capability Maturity Model Integration]



Software Engineering Institute, CMU, 2002
http://www.sei.cmu.edu/cmmi/

Apéndice: Modelos

Desarrollo ágil



Agile Project Management: Iteration

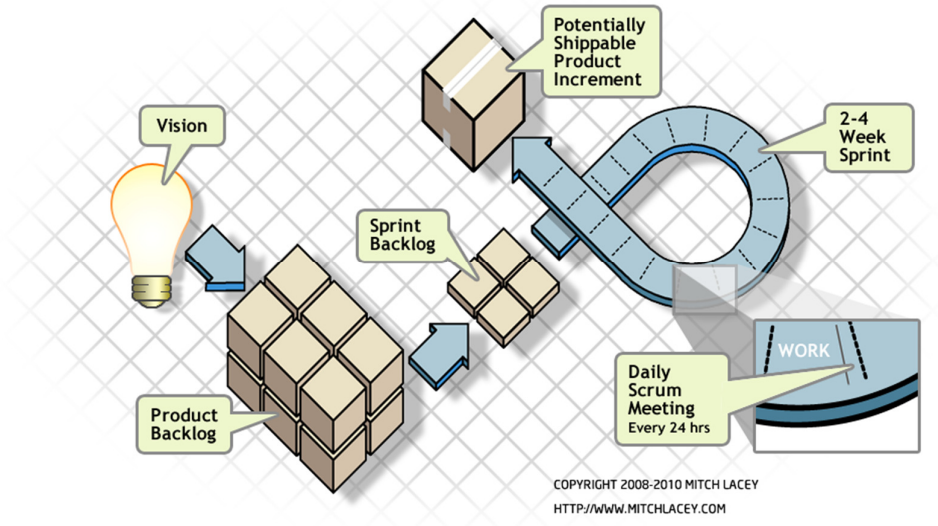
Apéndice: Modelos

XP [eXtreme Programming]

Kent Beck:
"Extreme Programming Explained: Embrace Change", 1999
ISBN 0201616416

Apéndice: Modelos

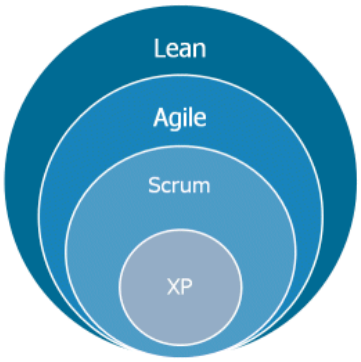
Scrum

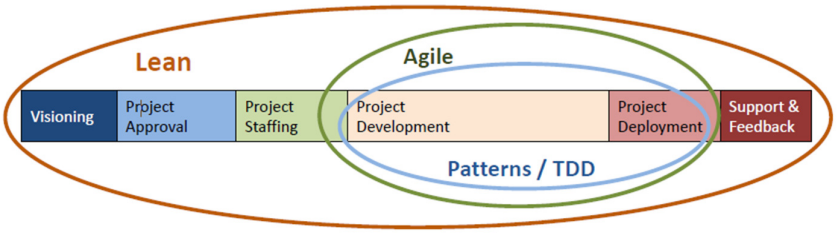


Jeffrey Sutherland & Ken Schwaber:
"Business object design and implementation"
OOPSLA'1995 workshop proceedings. ISBN 3-540-76096-2.

Apéndice: Modelos

LSD [Lean Software Development]

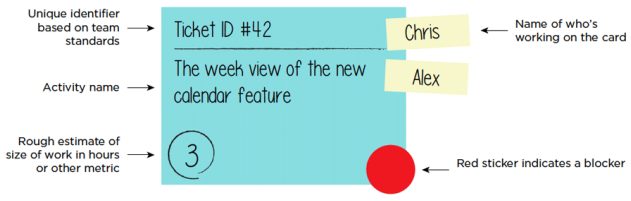


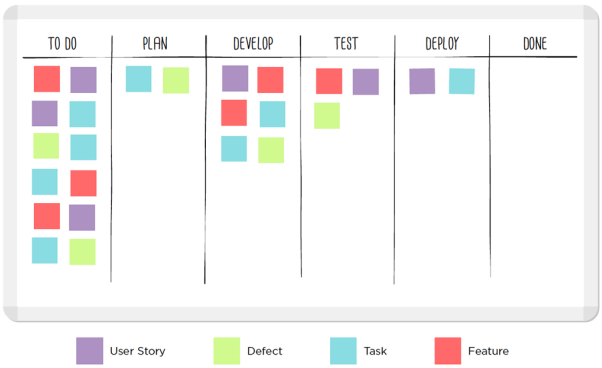


Mary Poppendieck & Tom Poppendieck:
"Software Development: An Agile Toolkit", 2003
ISBN 0-321-15078-3

Apéndice: Modelos

Kanban

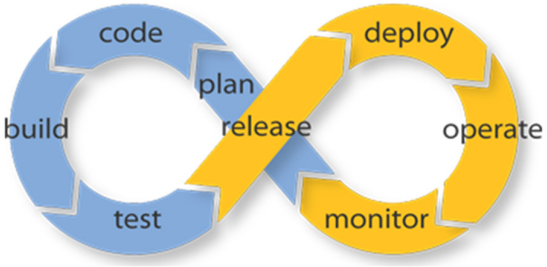




David Anderson:
"Agile Management for Software Engineering:
Applying the Theory of Constraints for Business Results", 2003.
ISBN 0-13-142460-2.

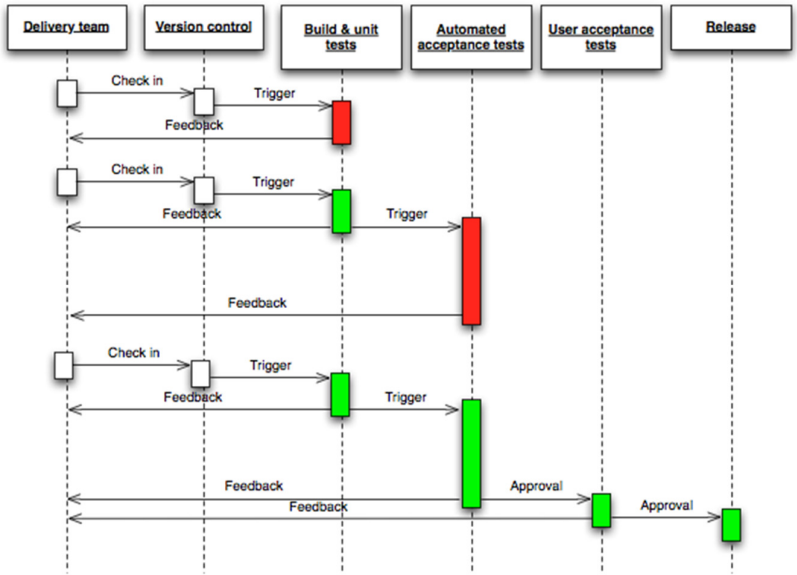
Apéndice: Modelos

CI [Continuous Integration]



Apéndice: Modelos

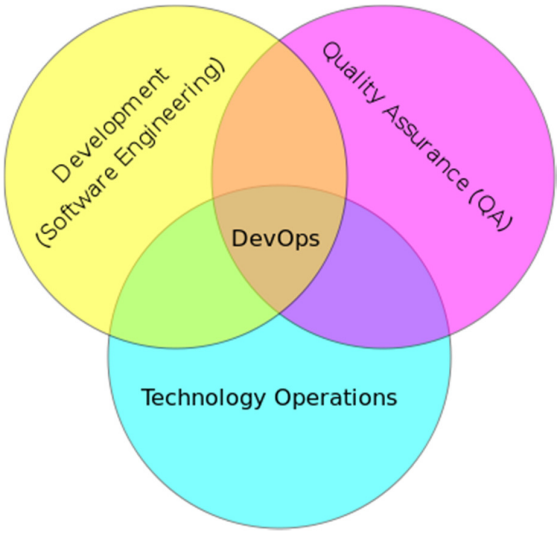
CD [Continuous Delivery]



Jez Humble & David Farley: "Continuous Delivery: Reliable Software Releases Through Build, Test and Deployment Automation", 2010. ISBN 978-0-321-60191-9.

Apéndice: Modelos

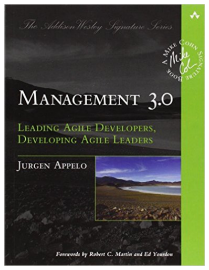
DevOps [Development + Operations]

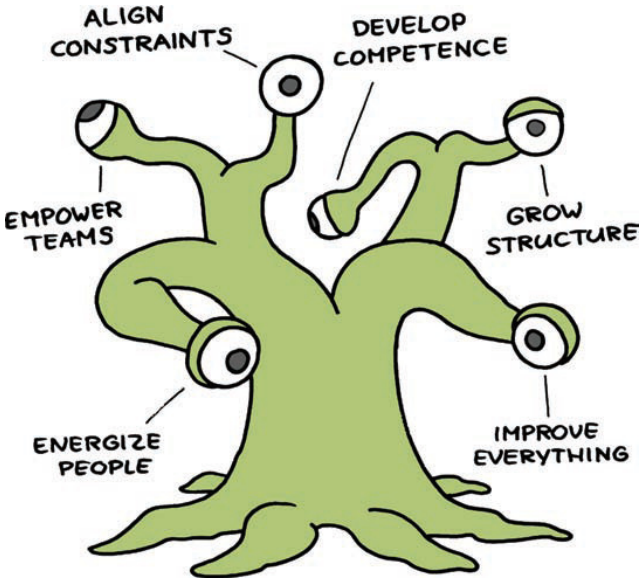


Andrew Clay Shafer & Patrick Debois:
"Agile Infrastructure", Agile 2008 Conference
http://www.jedi.be/presentations/agile-infrastructure-agile-2008.pdf

Apéndice: Modelos

Martie, el modelo de gestión 3.0





Apéndice: Modelos

The Toyota Way 2001

Respeto a las personas & Mejora continua

Los 14 principios de Deming

W. Edwards Deming: "Out of the Crisis", 1986

El modelo de 6 planos de Mintzberg

Henry Mintzberg: "Managing", 2009

Los 5 principios de Hamel

Gary Hamel: "The Future of Management", 2007

Apéndice: SWEBOK

Software Engineering Body of Knowledge

- Software Configuration Management
- Software Construction
- Software Design
- Software Engineering Management
- Software Engineering Process
- Software Maintenance
- Software Quality
- Software Requirements
- Software Testing
- Software Tools and Methods

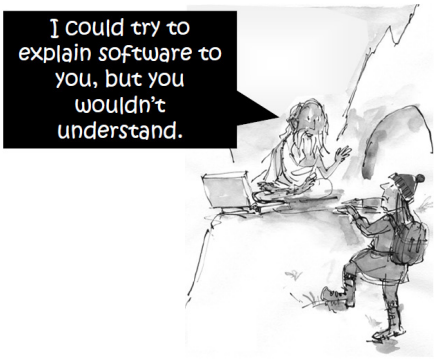
Apéndice: SWEBOK

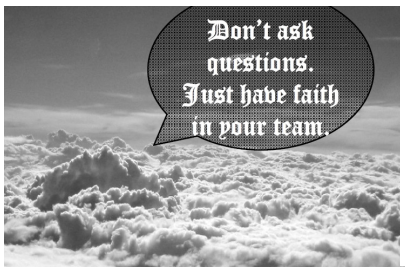
Software Engineering Body of Knowledge

To organize something is to understand it. – *Aristotle*

Truth will sooner come out of error than from confusion. – *Francis Bacon*

Mejor que las alternativas…



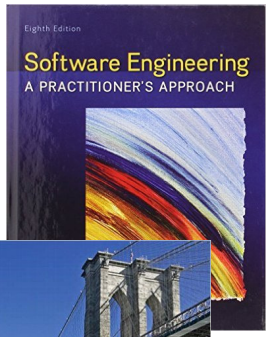


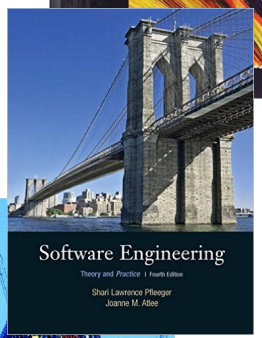
I could explain software to you, but I'd have to kill you.

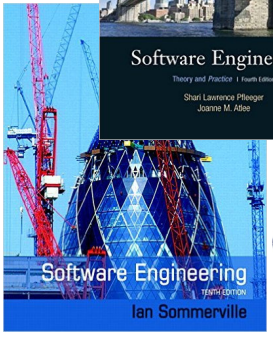
Steve McConnell (Construx Software):
"The 10 Most Important Ideas in Software Development"

Bibliografía

Libros de texto







- Roger S. Pressman:
  Software Engineering: A Practitioner's Approach
  McGraw-Hill, 8th edition, 2014. ISBN 0078022126
- Shari Lawrence Pfleeger & Hoanne M. Atlee:
  Software Engineering: Theory and Practice
  Prentice Hall, 4th edition, 2009. ISBN 0136061699
- Ian Sommerville:
  Software Engineering
  Pearson, 10th edition, 2015. ISBN 0133943038

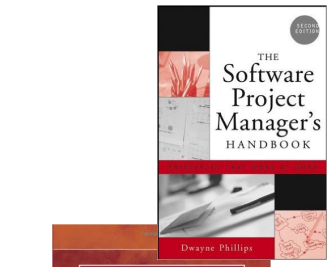
Bibliografía

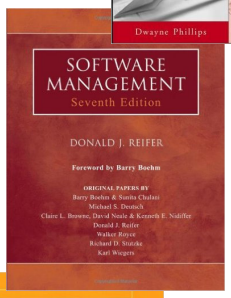
Lecturas recomendadas

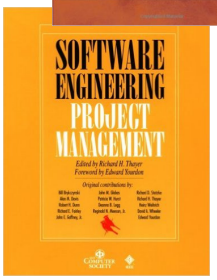
- Dwayne Phillips:
  The Software Project Manager's Handbook: Principles That Work at Work
  Wiley / IEEE Computer Society, 2nd edition, 2004
  ISBN 0471674206

- Donald J. Reifer (editor):
  Software Management
  Wiley / IEEE Computer Society, 7th edition, 2006
  ISBN 0471775622

- Richard H. Thayer (editor):
  Software Engineering Project Management
  Wiley / IEEE Computer Society, 2nd edition, 2000
  ISBN 0818680008



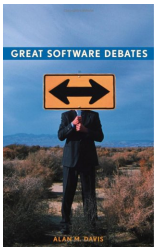


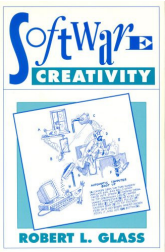


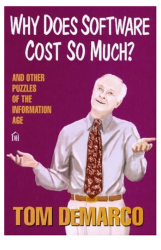
Bibliografía complementaria

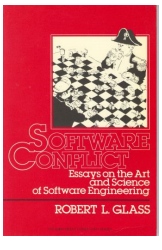
Ensayos

- Alan M. Davis:
  Great Software Debates
  Wiley - IEEE Computer Society Press, 2004.
  ISBN 0471675237
- Robert L. Glass:
  Software Creativity
  Prentice-Hall, 1995. ISBN 0131473646
- Robert L. Glass:
  Software Conflict: Essays on the Art and Science of Software Engineering
  Yourdon Press, 1990. ISBN 0138261571
- Tom DeMarco:
  Why does software cost so much? and other puzzles of the Information Age
  Dorset House, 1995. ISBN 093263334X









Bibliografía complementaria

Gestión del proceso de desarrollo de software

- Peter DeGrace & Leslie Hulet Stahl:
  Wicked Problems, Righteous Solutions:
  A Catalogue of Modern Software Engineering Paradigms
  Yourdon Press, 1990. ISBN 013590126X
- Jim McCarthy:
  Dynamics of Software Development
  Microsoft Press, 2006. ISBN 0735623198
- Michael Feathers:
  Working Effectively with Legacy Code
  Prentice-Hall PTR, 2004. ISBN 0131177052
- Cem Kaner, James Bach & Bret Pettichord:
  Lessons learned in software testing
  Wiley Computer Publishing, 2002. ISBN 0471081124

Bibliografía complementaria

Gestión del proceso de desarrollo de software

- Craig Larman:
  Agile and Iterative Development: A Manager's Guide
  Addison-Wesley Professional, 2003. ISBN 0131111558
- Robert C. Martin:
  Agile Software Development:
  Principles, Patterns, and Practices
  Prentice-Hall, 2003. ISBN 0135974445
- Alistair Cockburn:
  Agile Software Development
  Addison-Wesley, 2002. ISBN 0201699699
- James A. Highsmith III:
  Adaptive Software Development:
  A collaborative approach to managing complex systems
  Dorset House, 2001. ISBN 0932633404

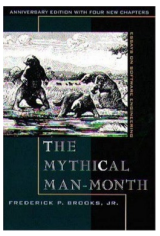
Bibliografía complementaria

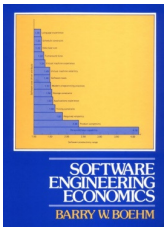
Gestión del proceso de desarrollo de software

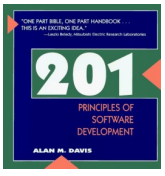
- Walker Royce:
  Software Project Management: A Unified Framework [RUP]
  Addison-Wesley Professional, 1998. ISBN 0201309580
- Watts S. Humphrey:
  A Discipline for Software Engineering [PSP]
  Addison-Wesley Professional, 1994. ISBN 0201546108
- Watts S. Humphrey:
  Managing the Software Process [CMM]
  Addison-Wesley Professional, 1989. ISBN 0201180952
- Tom Gilb:
  Principles of Software Engineering Management [EVO]
  Addison-Wesley, 1988. ISBN 0201192462
- **Recommended Approach to Software Development**
  NASA Software Engineering Laboratory, SEL-81-305, rev.3, 1992

Bibliografía complementaria

Clásicos







- Frederick P. Brooks, Jr.:
  The Mythical Man-Month:
  Essays on Software Engineering
  Addison-Wesley, 1995. ISBN 0201835959
- Alan M. Davis:
  201 Principles of Software Development
  McGraw-Hill, 1995. ISBN 0070158401
- Barry W. Boehm:
  Software Engineering Economics
  Prentice-Hall PTR, 1991. ISBN 0138221227
- **Manager's Handbook for Software Development**
  NASA Software Engineering Laboratory, SEL-84-101, rev.1, 1990.
- **Software Engineering Laboratory (SEL)**
  Relationships, Models, and Management Rules
  NASA Software Engineering Laboratory, SEL-91-001, 1991.

Bibliografía

Bibliografía en castellano

- Roger S. Pressman:
  Ingeniería de Software: Un enfoque práctico
  McGraw-Hill, 7ª edición, 2010. ISBN 6071503140
- Ian Sommerville:
  Ingeniería de Software
  Pearson, 9ª edición, 2012. ISBN 6073206038