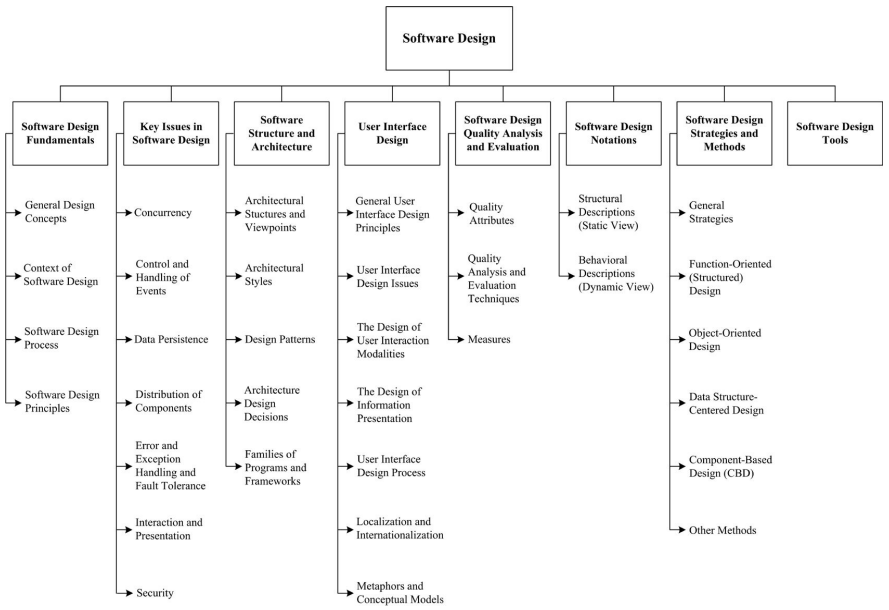
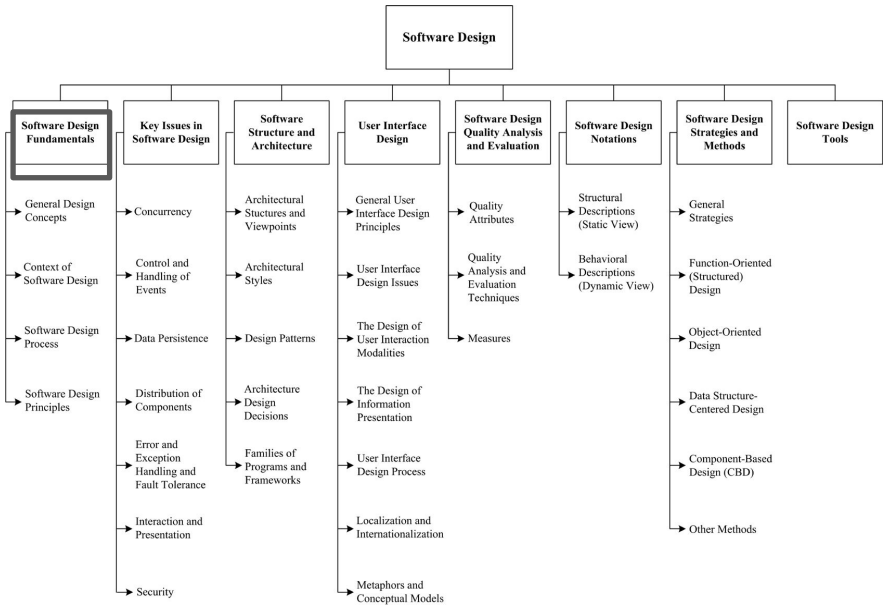
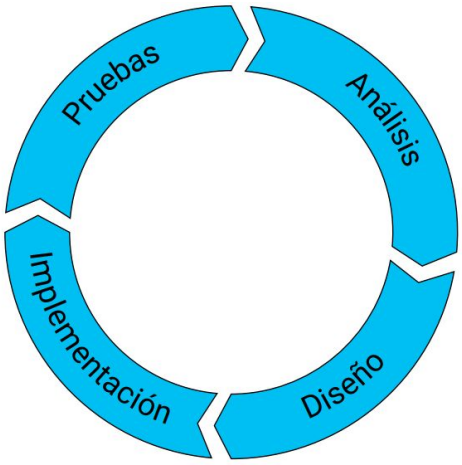
Diseño de software

Sivana Hamer - sivana.hamer@ucr.ac.cr
Escuela de Ciencias de la Computación





El diseño de software es parte de la resolución de problemas



Para obtener los requerimientos del diseño, vamos a usar historias de usuario

User Story

As a <role>
I want <goal>
so that <benefit>

Acceptance criteria:
(Conditions of Satisfaction)
…
…

As an Account Manager
I want a sales report of my account to be sent to my inbox daily
So that I can monitor the sales progress of my customer portfolio

Acceptance criteria:
1. The report is sent daily to my inbox
2. The report contains the following sales details: …
3. The report is in csv format.

¿Para los problemas de software existe una solución definitiva?

Si

No

Poll locked. Responses not accepted.

¿Para los problemas de software existe una solución definitiva?

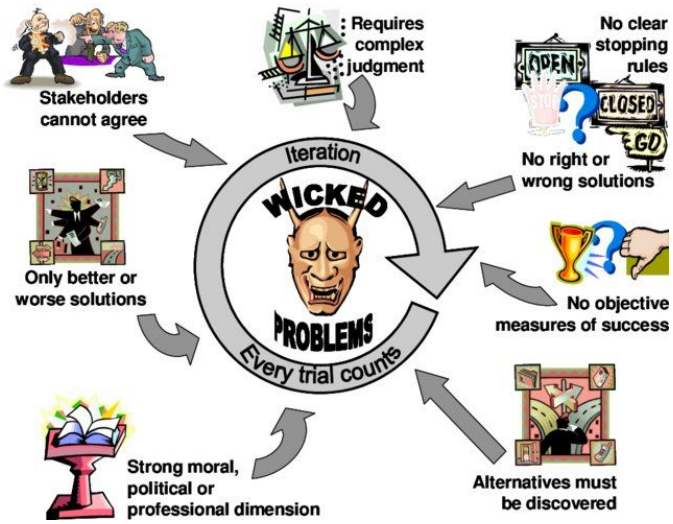
Si

No

Powered by Poll Everywhere

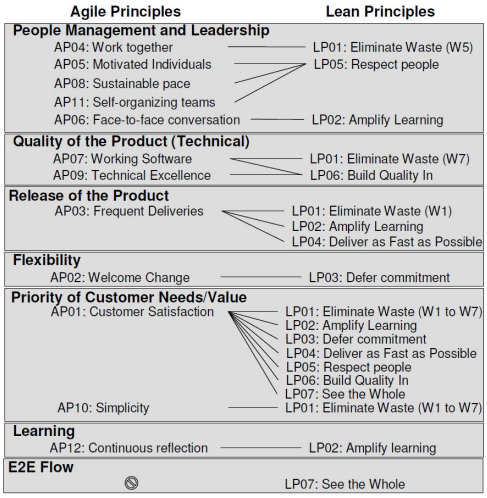
Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

En software, generalmente no hay una solución definitiva



Maqsood

Principios (*principles*) son reglas que se deben seguir para alcanzar objetivos



Petersen

Existen distintos principios en la ingeniería de software, aunque vamos a enfocarnos en la etapa de diseño

DRY **OOP** **Otros**

KISS **SOLID**

When poll is active, respond at **pollev.com/sivanahamer104**

¿Cuáles de estos principios conocen?

DRY

KISS

SOLID

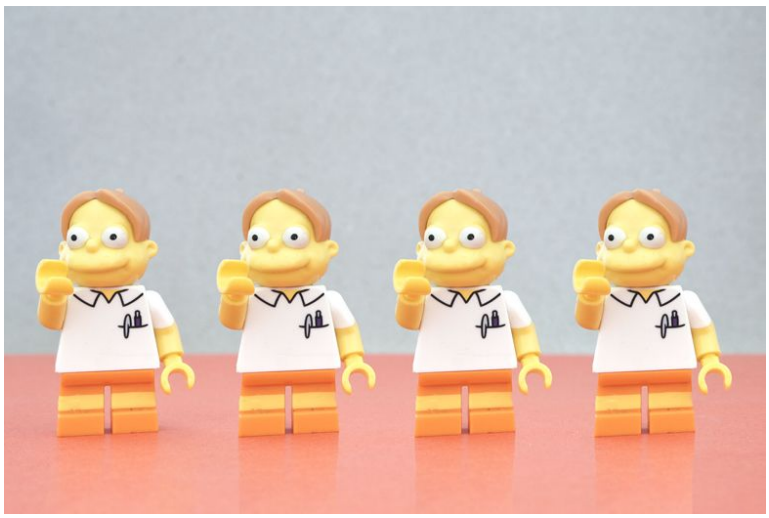
OOP

Ninguno

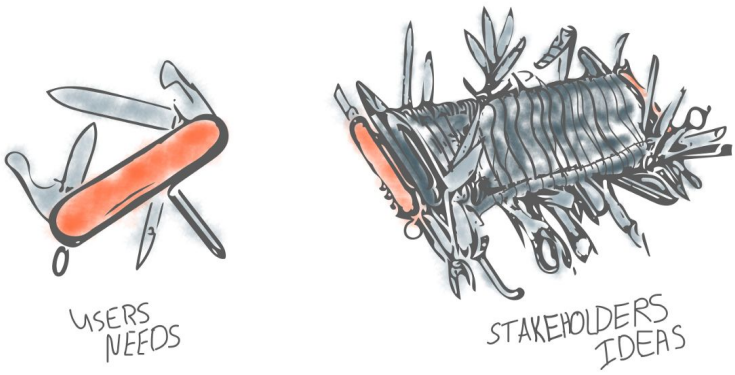
Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Don't repeat yourself (DRY) busca en reducir la duplicidad en software



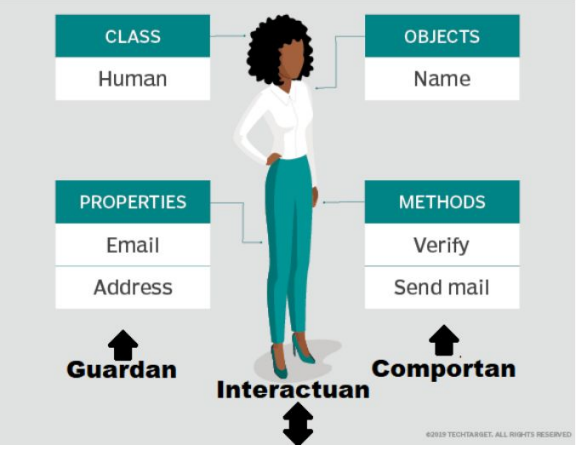
Keep it simple stupid (KISS) busca que los sistemas sean lo más simples posibles



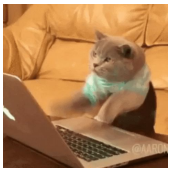
"Everything Should Be Made as Simple as Possible, But Not Simpler." Einstein?

https://quoteinvestigator.com/2011/05/13/einstein-simple/

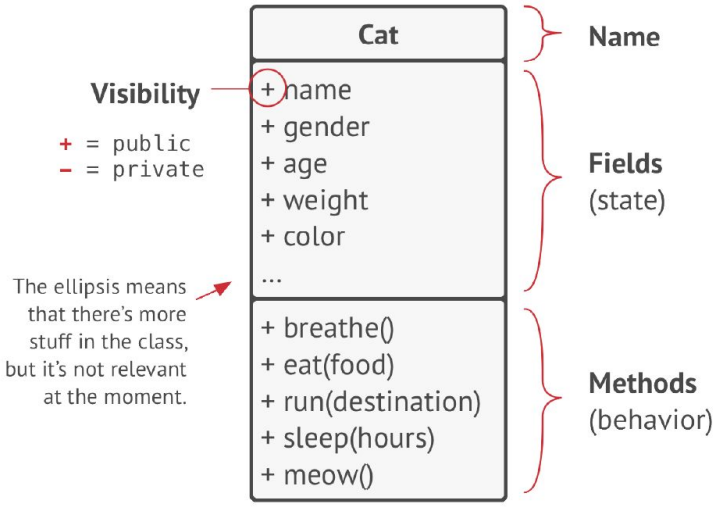
Programación orientada a objetos (OOP) nos enfocamos en los objetos



Vamos a hablar de gatos…



En UML una clase de gato se representa como…



Shvets

De una clase, se pueden instanciar varios objetos

Oscar: Cat

```
name    = "Oscar"
sex     = "male"
age     =  3
weight  =  7
color   =  brown
texture =  striped
```

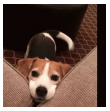
Luna: Cat

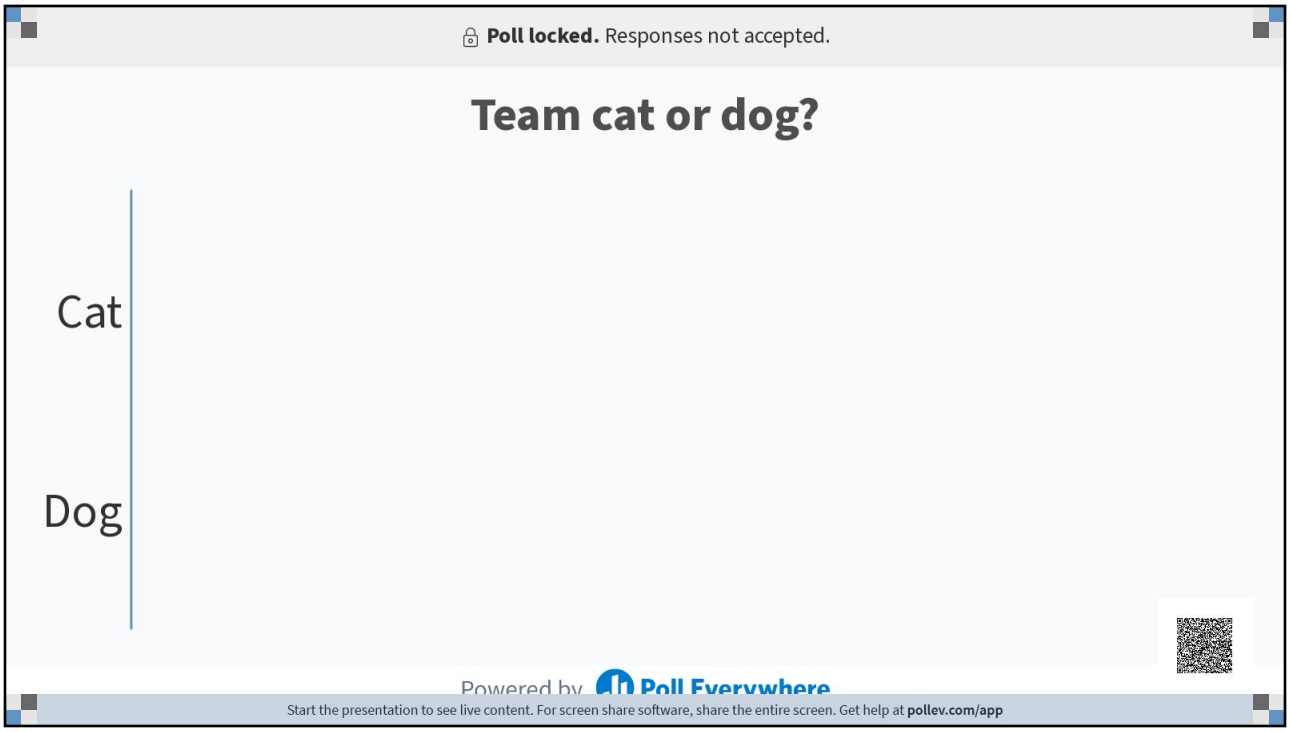
```
name    = "Luna"
sex     = "female"
age     =  2
weight  =  5
color   =  gray
texture =  plain
```

Shvets

Además, hablemos de perros…



Poll locked. Responses not accepted.

Team cat or dog?

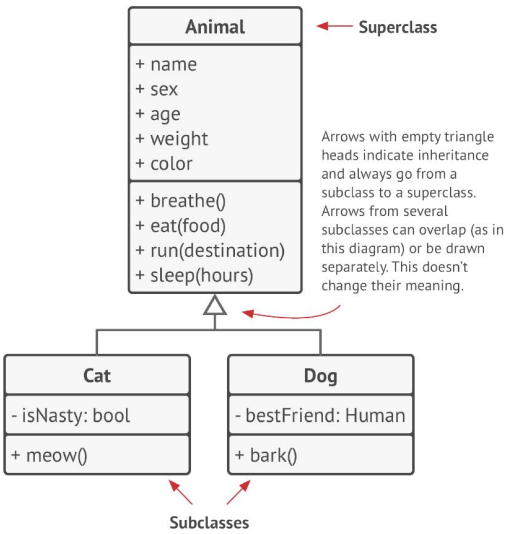
Cat

Dog

Powered by Poll Everywhere

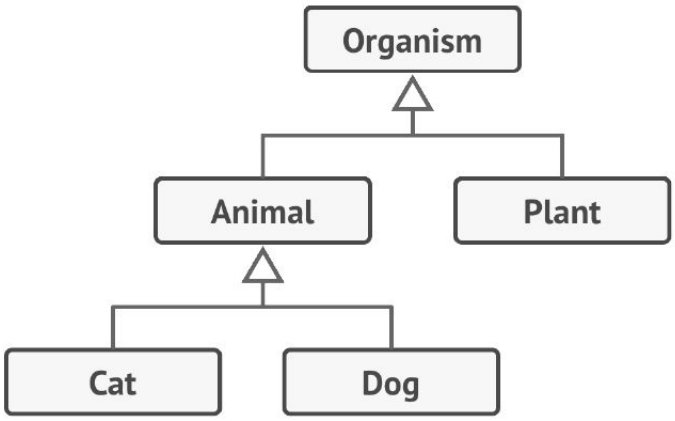
Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

Se puede crear una superclase de animales para heredar



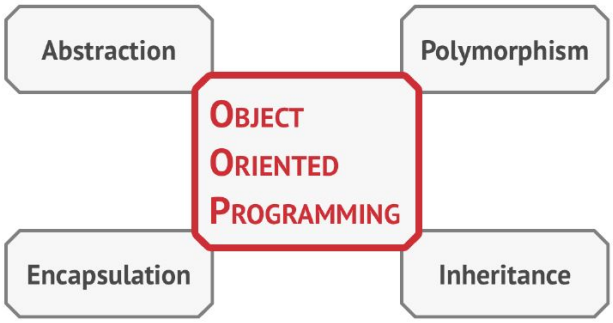
Shvets

Se puede abstraer más las clases



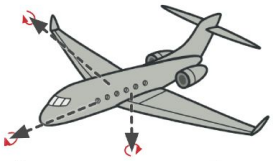
Shvets

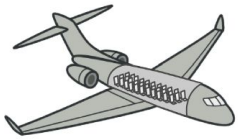
En la programación orientado a objetos, hay varios pilares o principios de diseño

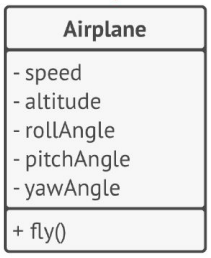


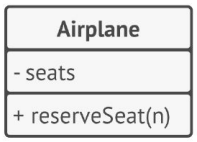
Shvets

Al abstraer, modelos ciertos aspectos de una clase pero no es una representación 100% preciso



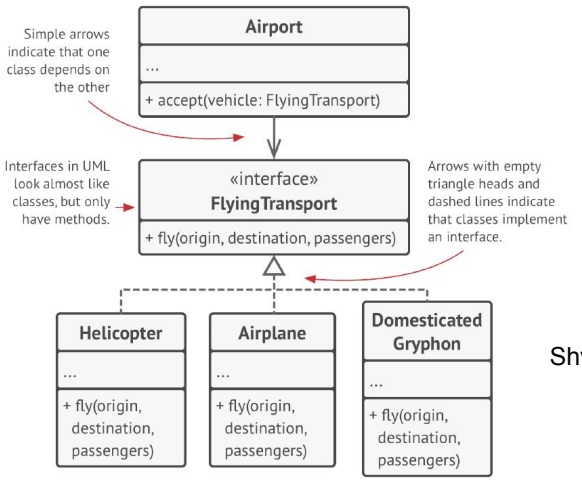






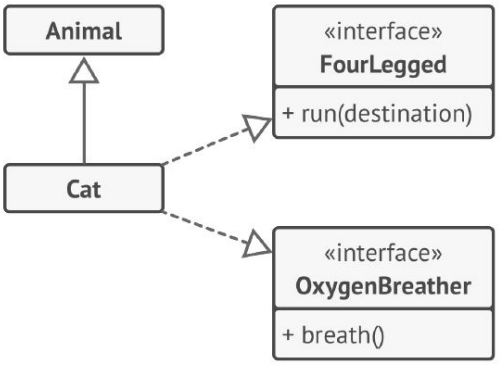
Shvets

Al encapsular una implementación se ocultan estados y comportamientos, limitándose al resto del programa.



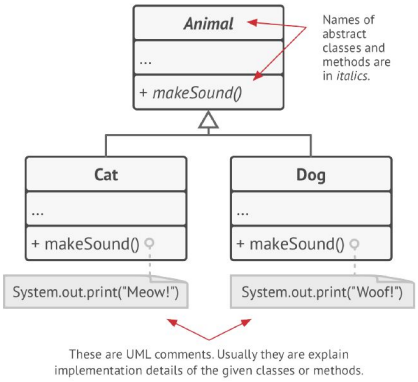
Shvets

Al heredar clases se puede reusar código de otras clases y extender la funcionalidad.



Shvets

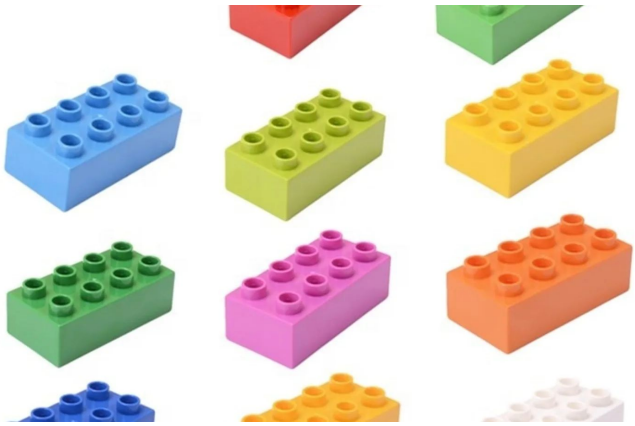
El polimorfismo es la habilidad de una subclase de tener su propia implementación siendo parte de una clase



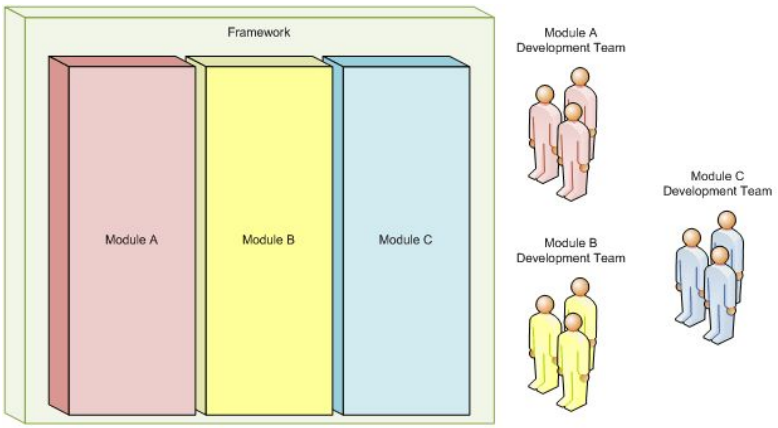

```
1  bag = [new Cat(), new Dog()];
2
3  foreach (Animal a : bag)
4    a.makeSound()
5
6  // Meow!
7  // Woof!
```

Shvets

Separation of concerns (SoC) se puede subdividir cada problema en subproblemas con preocupaciones distintas reduciendo complejidad



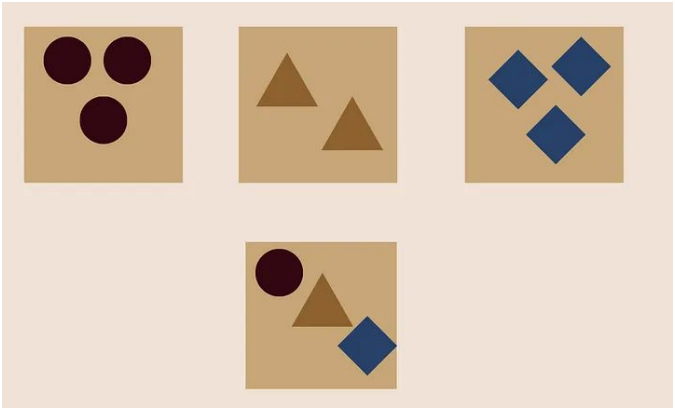
Modularizar o descomponer es cuando se hace Soc al dividir el software en módulos.



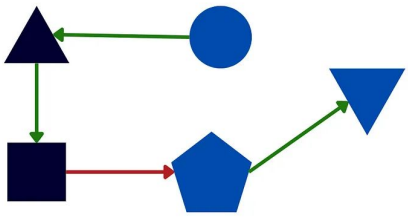
La cohesión indica el grado en que un componente, módulo o clase se enfoca en hacer solo una cosa.

Alto

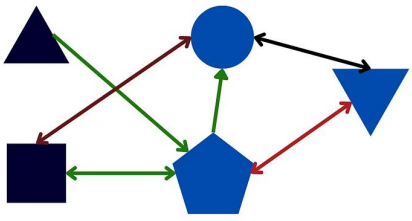
Bajo



El acoplamiento indica el grado en que un componente, módulo o clase está interconectado con otros

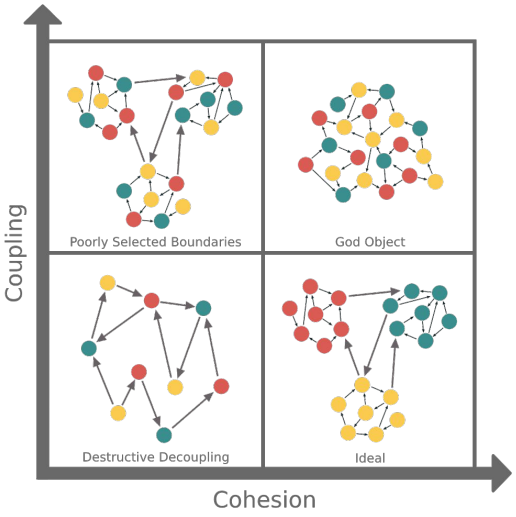


Bajo

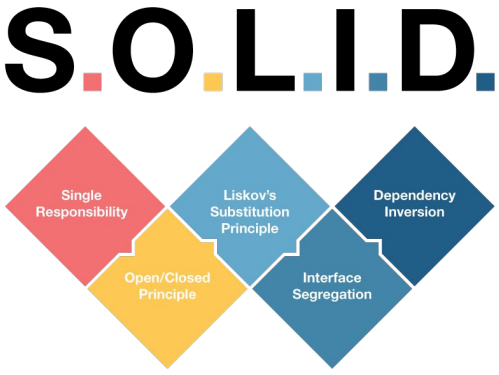


Alto

Buscamos alta cohesión y bajo acoplamiento



SOLID es un acrónimo de una serie de prácticas para hacer el código adaptable al cambio.

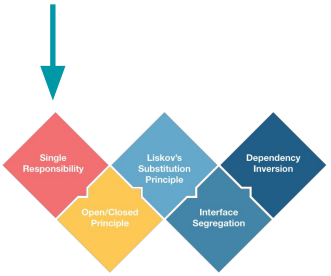


Robert Martin

(uncle bob, es toda)

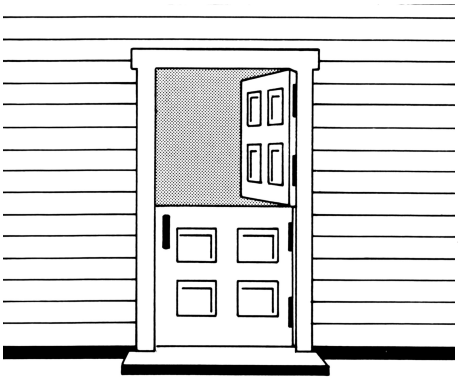
Ingeniero de software parte del manifiesto ágil. Ha escrito mucho sobre el tema de código y arquitecturas limpias. También ha trabajado en patrones de diseño, SOLID, XP y TDD.

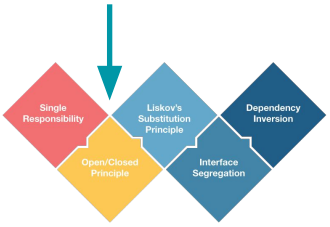
Cada clase debería tener una responsabilidad única



LEARN stuff

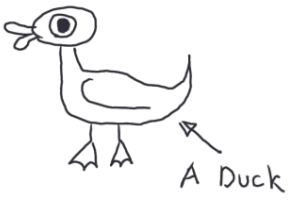
El código está abierto a extender pero cerrado al cambio

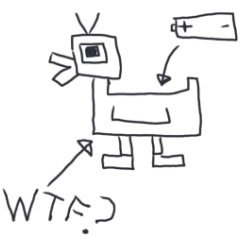


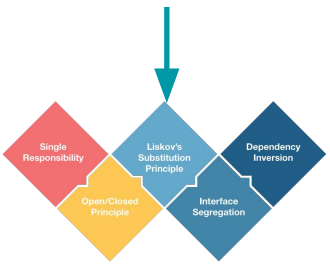


Hay excepciones de este principio

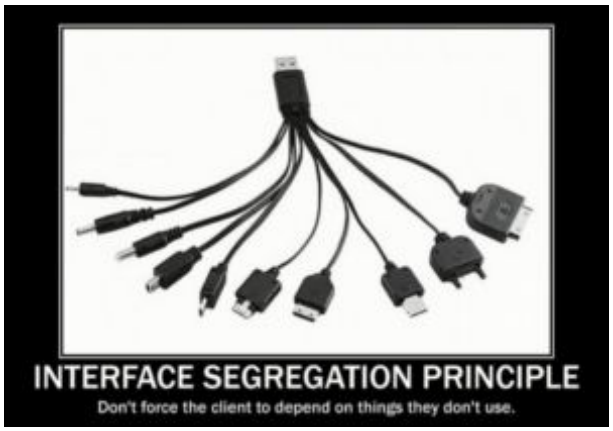
Cada objeto de una superclase puede ser cambiado por un objeto de una subclase sin romper la aplicación

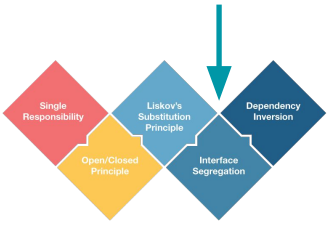




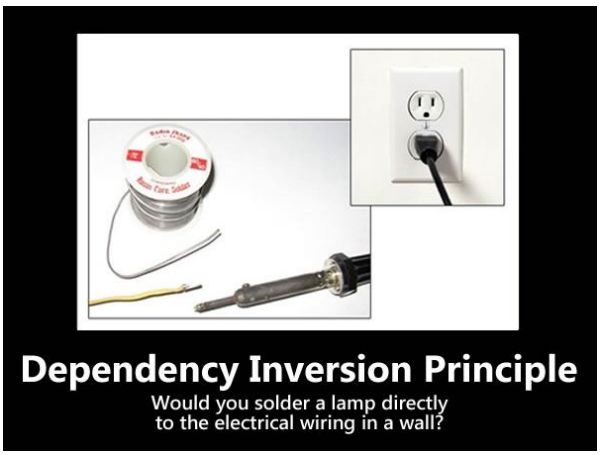


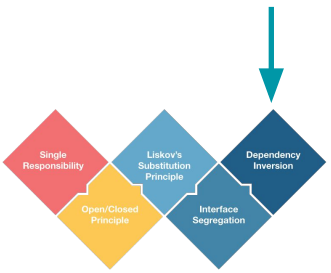
Clases no deberían ser forzados de depender de métodos que no usan



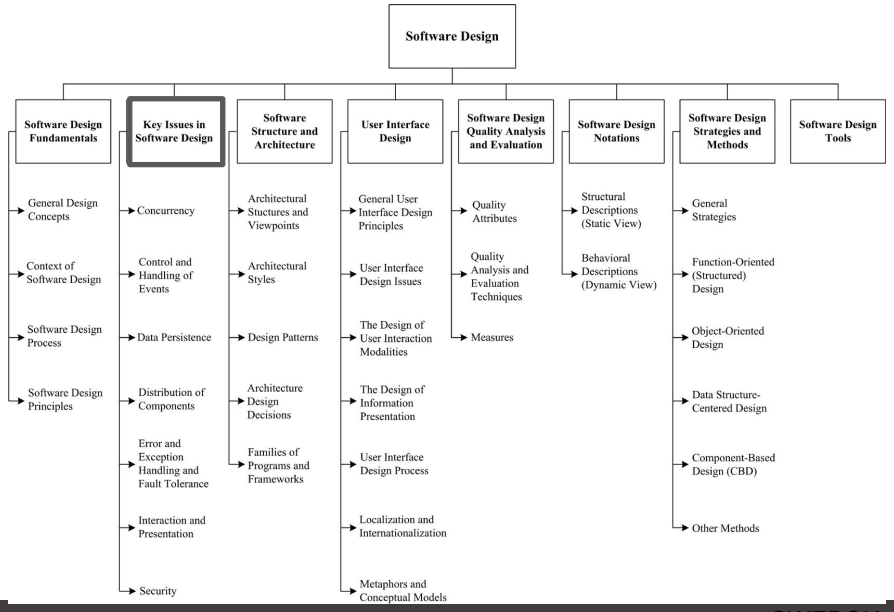


Se depende de abstracciones, no implementaciones



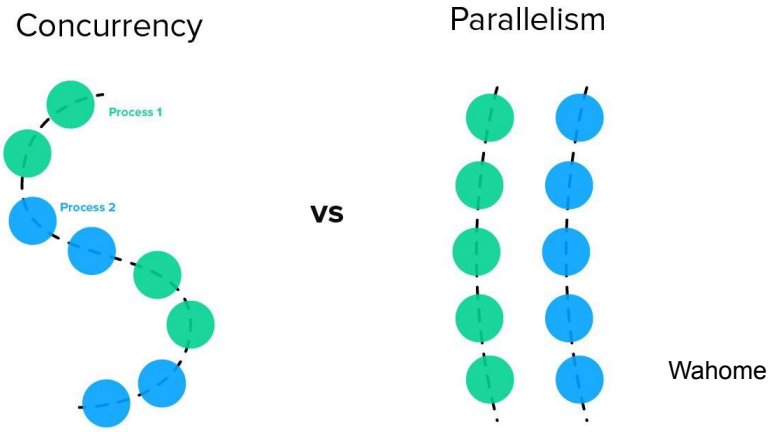


Veamos ejemplos de la mayoría de principios SOLID :)



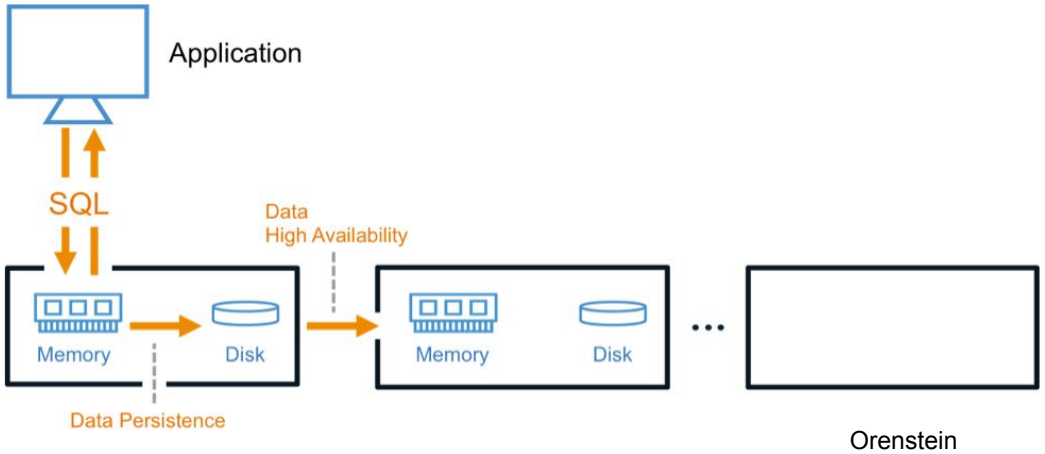
SWEBOK

Se considera la concurrencia, incluyendo la atomicidad, sincronización, eficiencia y *scheduling*



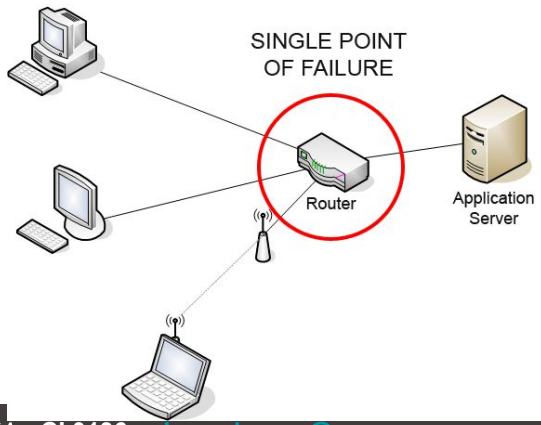
Se considera el manejo el manejo y control de eventos a través de eventos

Se considera la persistencia de los datos

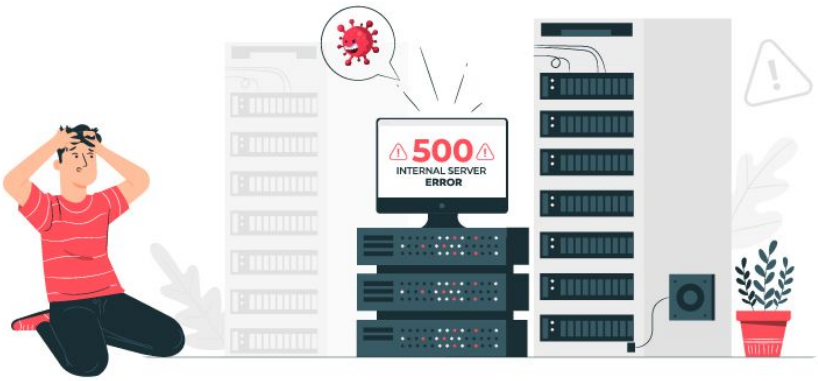


Orenstein

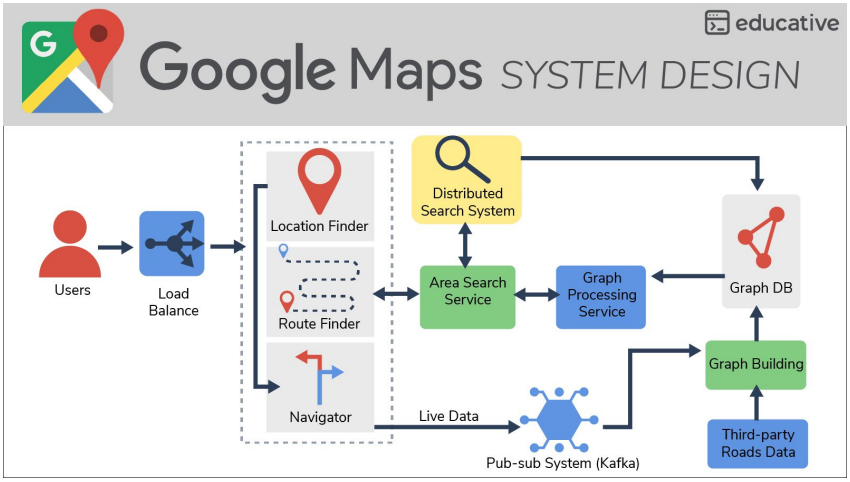
Se considera como se puede distributor el software en el hardware y comunican los componentes



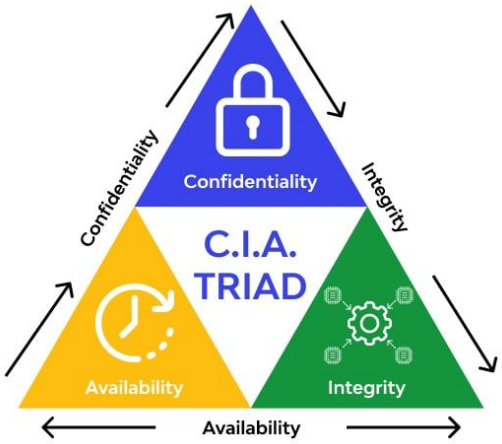
Se considera como manejar errores y ser tolerante a fallos

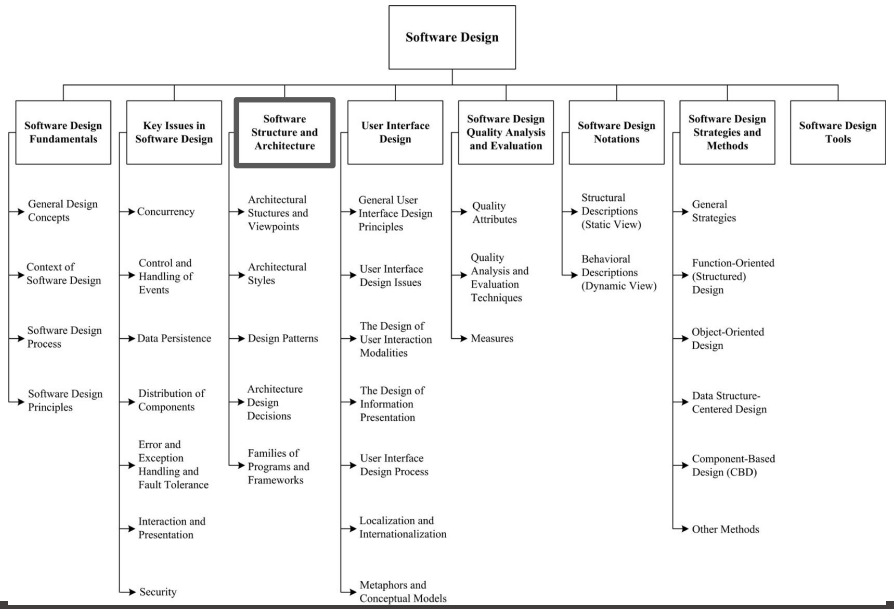


Se considera como se estructura las interacciones entre componentes de un sistema



Se considera como se puede reducir problemas de seguridad dentro de un sistema





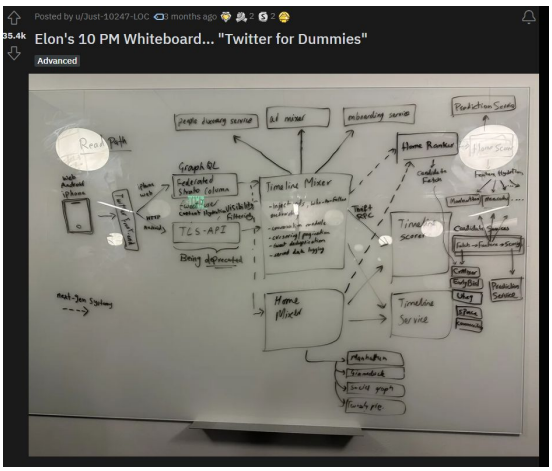
SWEBOK

¿Qué arquitecturas de software han escuchado anteriormente?

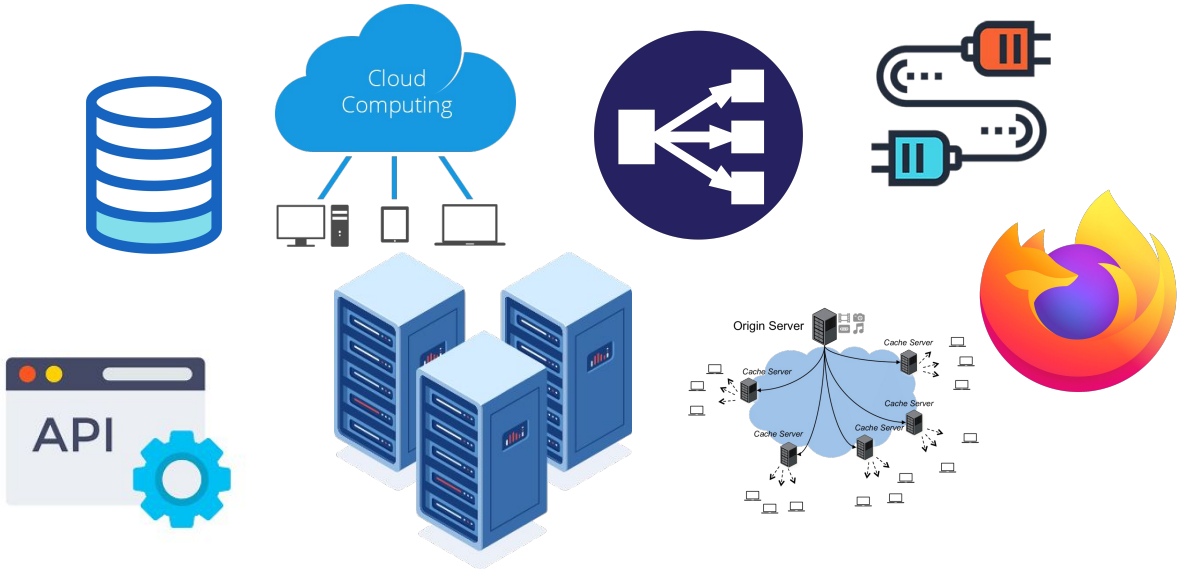
Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

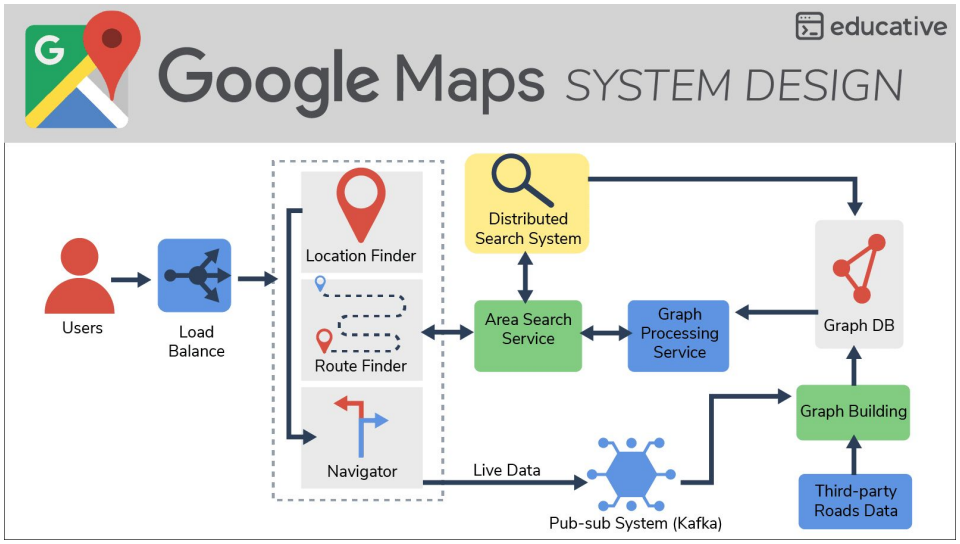
La arquitectura de un sistema representa los componentes con sus interacciones en un sistema



Un componente software es una parte del sistema



Ejemplo simplificado de google maps



Martin Fowler (es toda)

Ingeniero de software graduado de University College London. Popularizó la práctica de refactorización y participó en escribir el manifiesto ágil. Ha escrito nueve libros.

Software Architecture Guide

When people in the software industry talk about "architecture", they refer to a hazily defined notion of the most important aspects of the internal design of a software system. A good architecture is important, otherwise it becomes slower and more expensive to add new capabilities in the future.

Like many in the software world, I've long been wary of the term "architecture" as it often suggests a separation from programming and an unhealthy dose of pomposity. But I resolve my concern by emphasizing that good architecture is something that supports its own evolution, and is deeply intertwined with programming. Most of my career has revolved about the questions of what good architecture looks like, how teams can create it, and how best to cultivate architectural thinking in our development organizations. This page outlines my view of software architecture and points you to more material about architecture on this site.

A guide to material on martinfowler.com about software architecture.

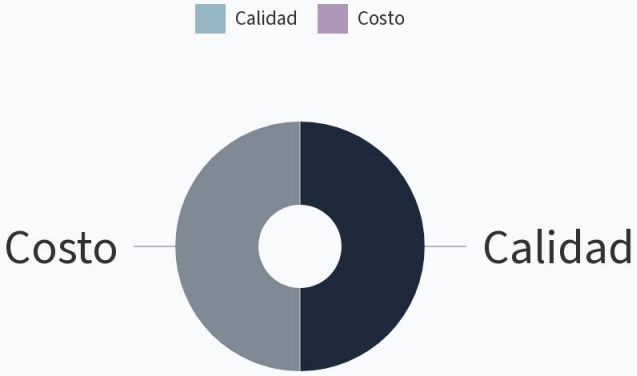
Martin Fowler

1 Aug 2019

https://martinfowler.com/architecture/

When poll is active, respond at **pollev.com/sivanahamer104**

¿Que importa más en el software?

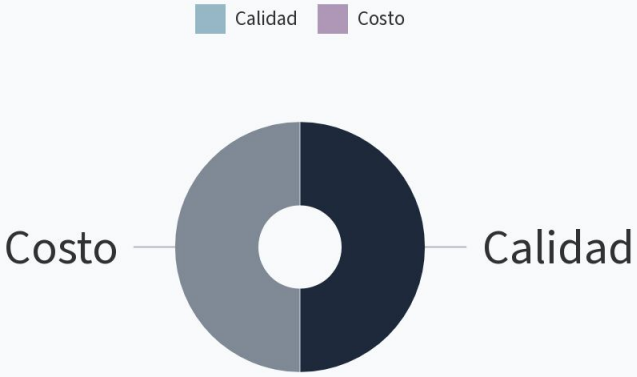


Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Poll locked. Responses not accepted.

¿Que importa más en el software?

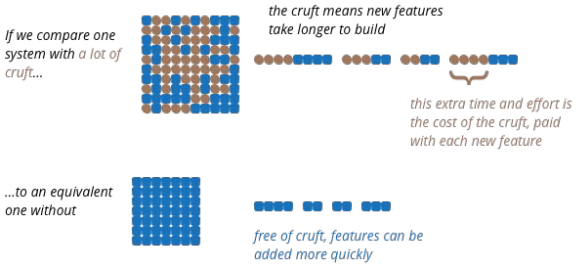


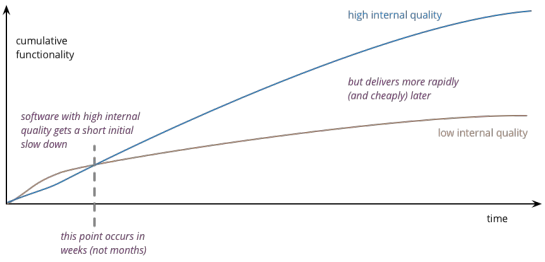
Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Al desarrollar software, siempre tenemos que balancear la calidad y el costo.

El punto de tener una arquitectura limpia ayuda a la productividad de desarrollar un sistema.





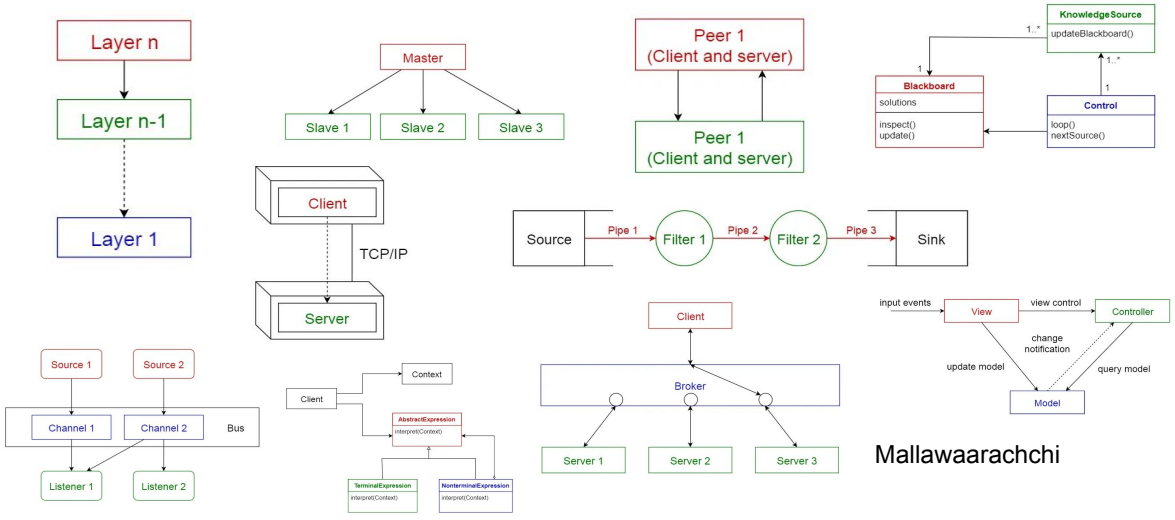
Fowler

Ward Cunningham

Ingeniero de software de Purdue University. Creó el primer wiki y es uno de los autores del manifiesto ágil. Pionero de testing, smalltalk y diseño.

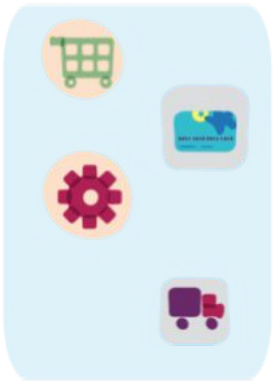
"*The best way to get the right answer on the Internet is not to ask a question; it's to post the wrong answer.*" Ward Cunningham

Existen muchas arquitecturas de software…



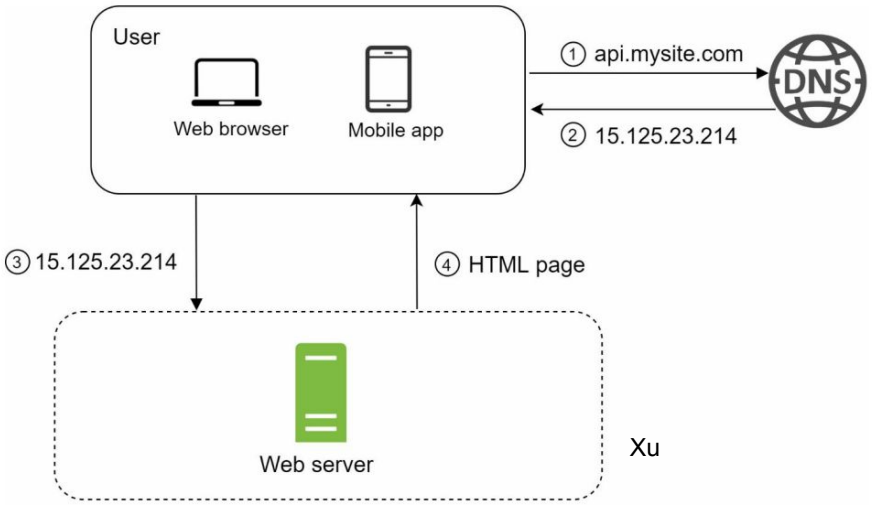
Mallawaarachchi

El patrón de software monolítico todo los componentes están dentro de un mismo servidor



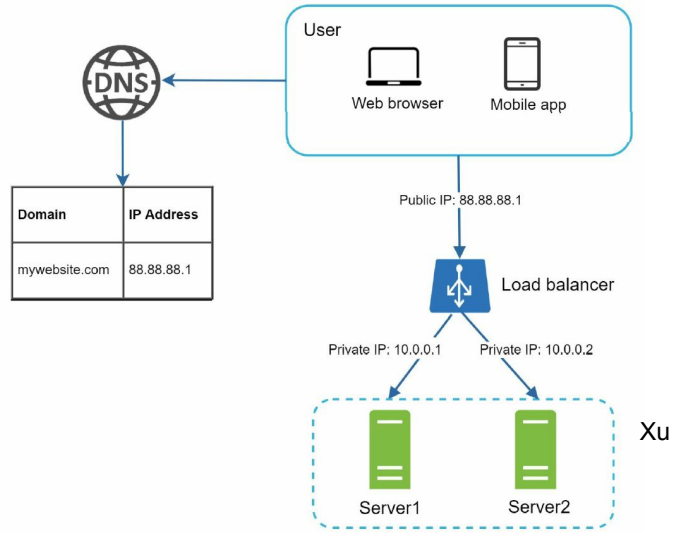
Fowler

Ejemplo de un servidor en acción



Xu

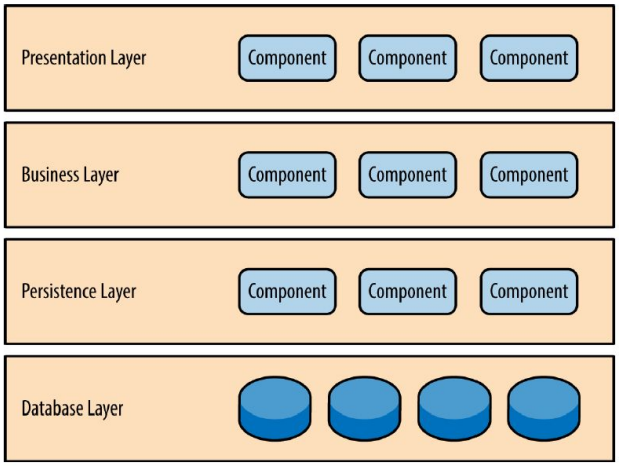
Ejemplo de manejar cargas…



Xu

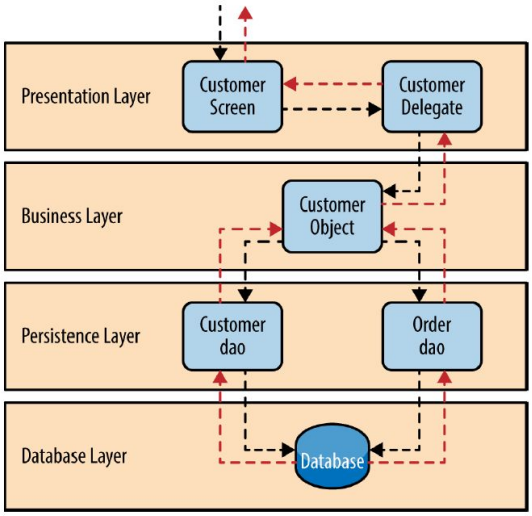
¿Qué sistema de la U les recuerda lo anterior?

En el multi capas, cada capa tiene algún rol dentro de la aplicación



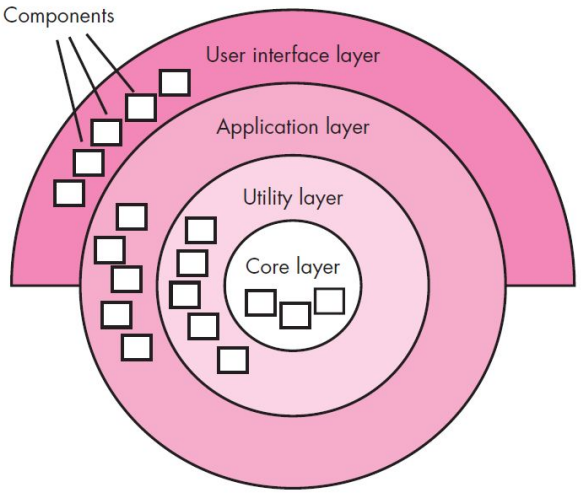
Richard

Un ejemplo de interacción en el multicapas…



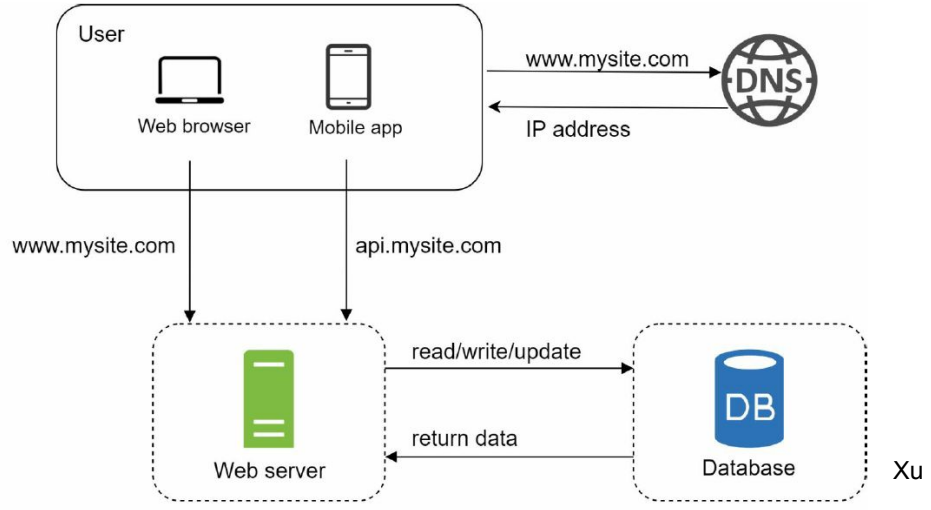
Richard

Se puede diseñar el multicapas de manera circular



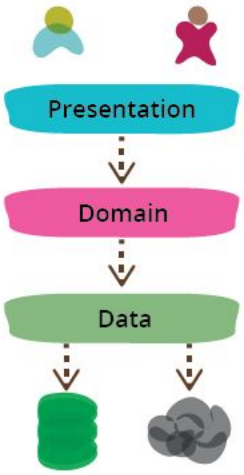
Pressman

Un ejemplo de arquitectura de dos capas…



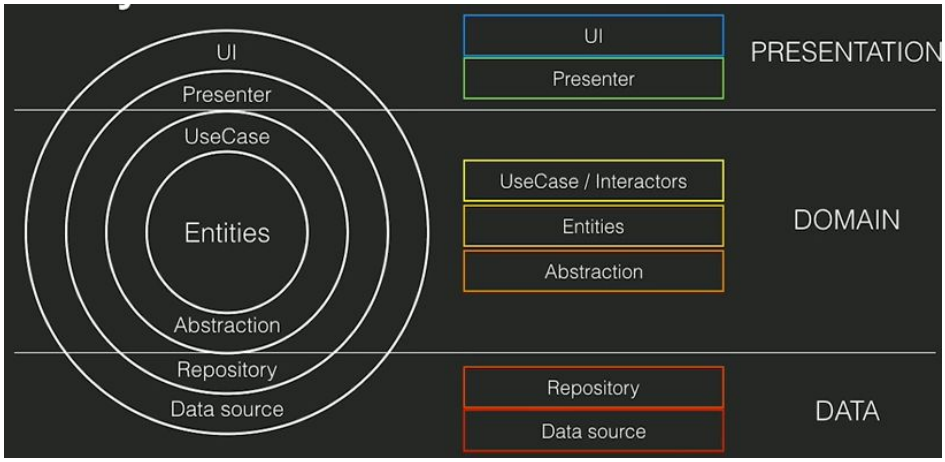
Xu

Es común utilizar tres capas…



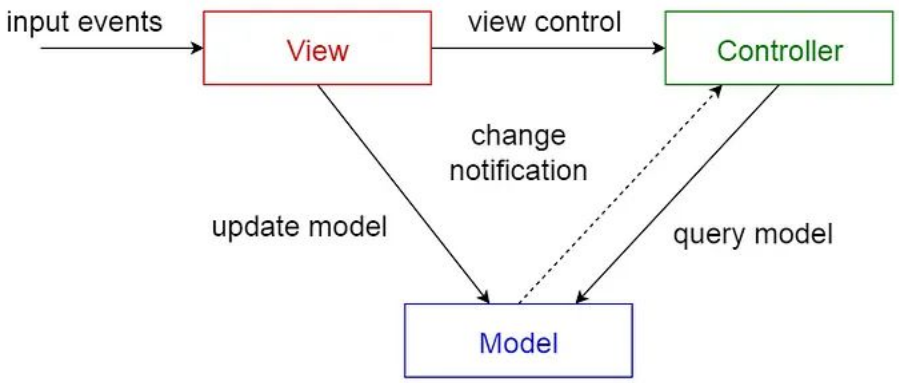
Fowler

En Domain Driven Design se divide el software en capas...



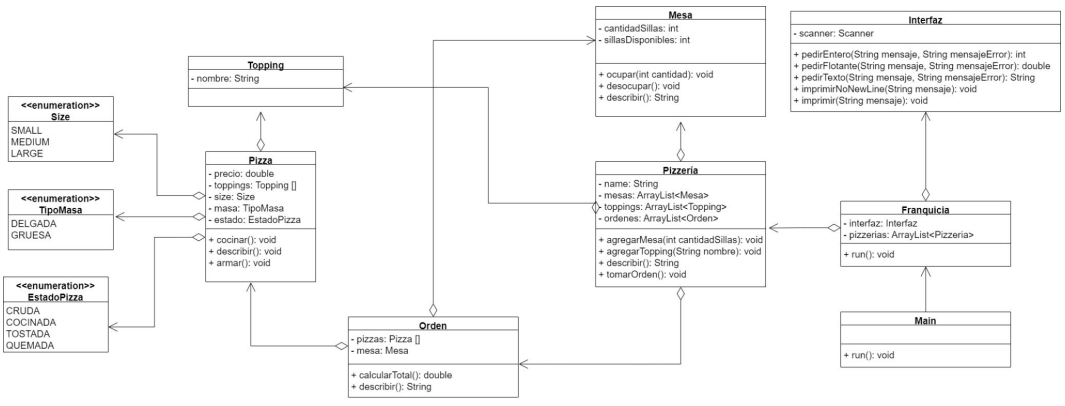
https://github.com/pycabook/rentomatic/tree/master Zimmerman

En el modelo vista controlador el sistema se divide en tres partes…



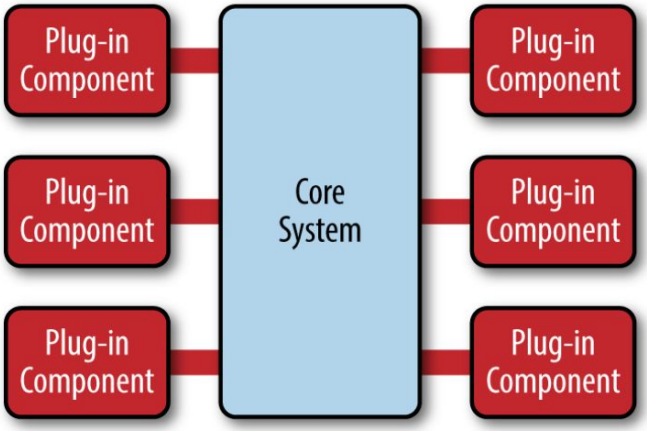
Mallawaarachchi

Ejemplo MVC de Java de una Pizzería



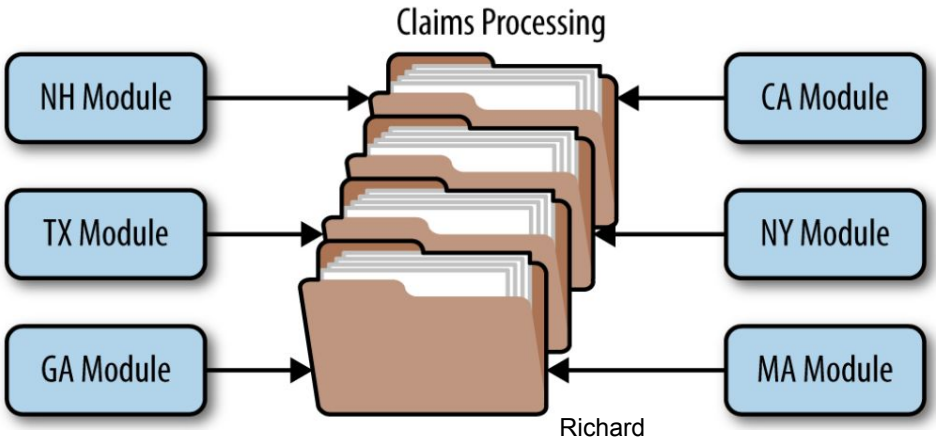
https://github.com/sivanahamer/programacion-1/tree/main/04-OOP/src/pizza

En el microkernel, hay un sistema core que se conecta con modelos plug-in.



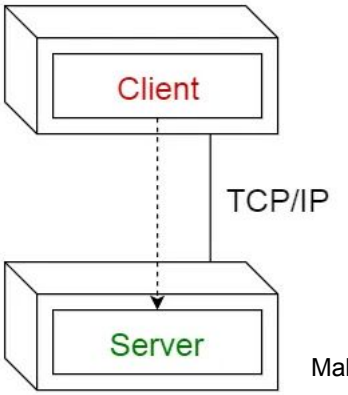
Richard

Un ejemplo de microkernel…



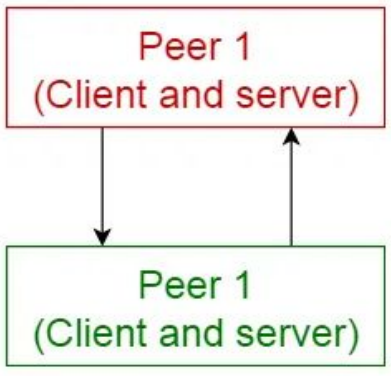
Richard

En el cliente server hay un cliente que se comunica con el server.



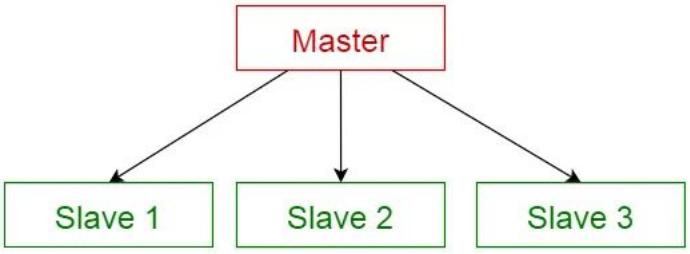
Mallawaarachchi

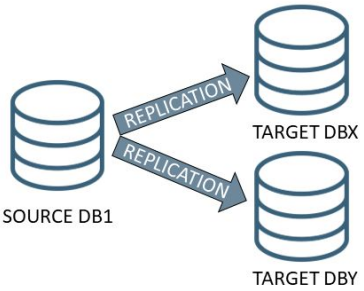
En peer to peer (p2p) cada componente actúa tanto como el cliente y servidor como *peers*.



Mallawaarachchi

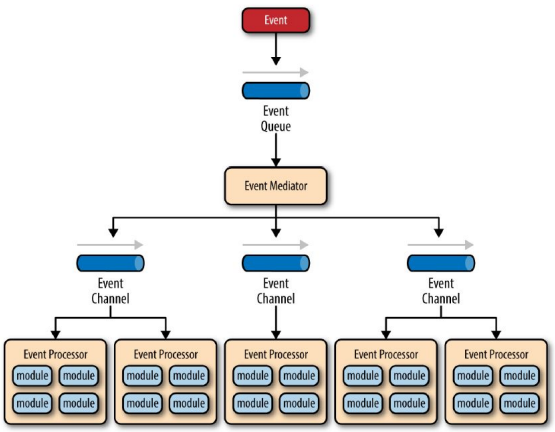
En maestro esclavo, el maestro distribuye el trabajo en componentes esclavos





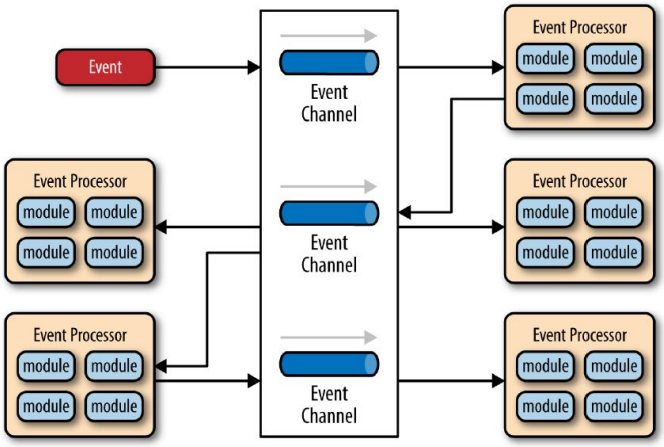
Richard

Una arquitectura basado en eventos permite recibir eventos asincrónicamente manejados por el mediador



Richard

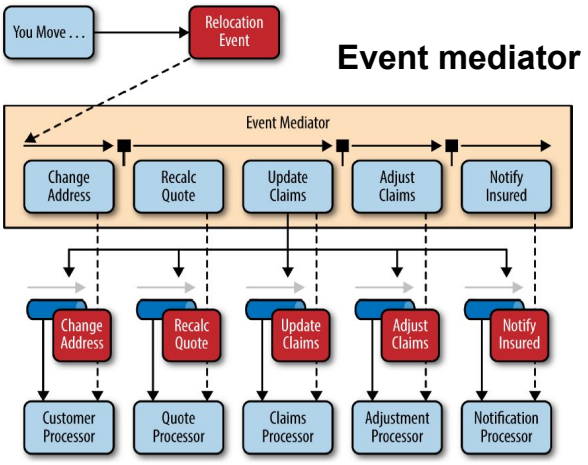
Con el patrón Broker, se encarga de coordinar comunicación entre componentes



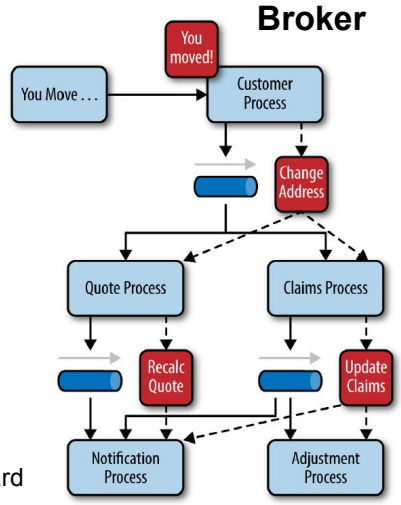
RabbitMQ

Richard

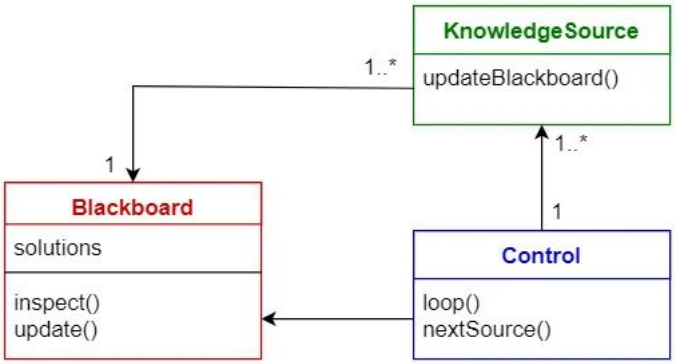
Diferencias entre mediador y broker…



Richard

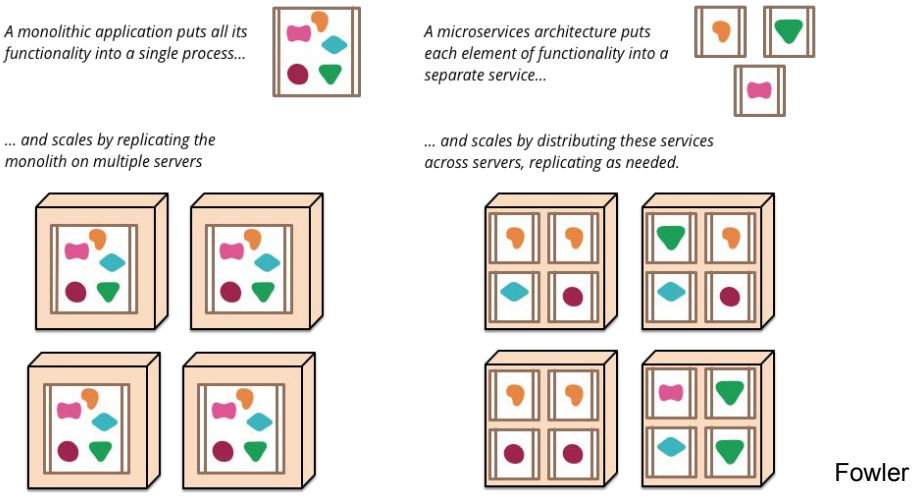


En una arquitectura blackboard el componente de control para leer el conocimiento y generar soluciones



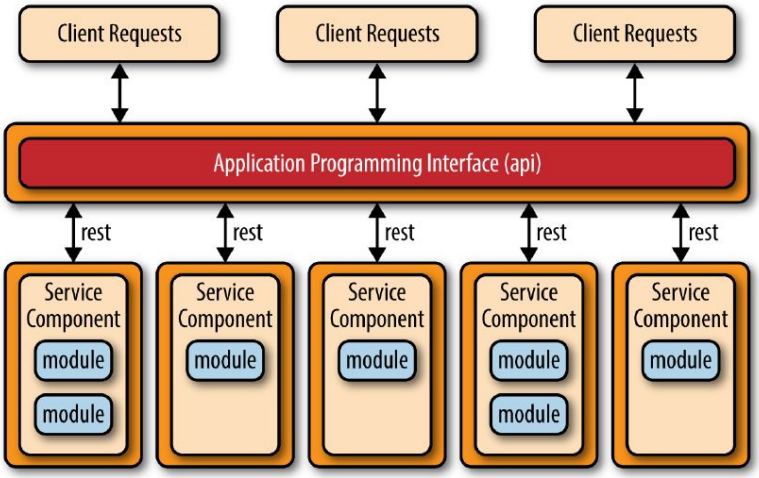
Mallawaarachchi

En los microservicios, se divide cada componente en un servicio



Fowler

Hoy en día es común el uso de APIs REST



Richard

¿Cuáles beneficios tiene una arquitectura de microservicios?

Diversidad de tecnologías
Consistencia eventual
Complejidad operacional
Sistemas distribuidos
Límites de módulos fuertes
Independicia al publicar (deploy)

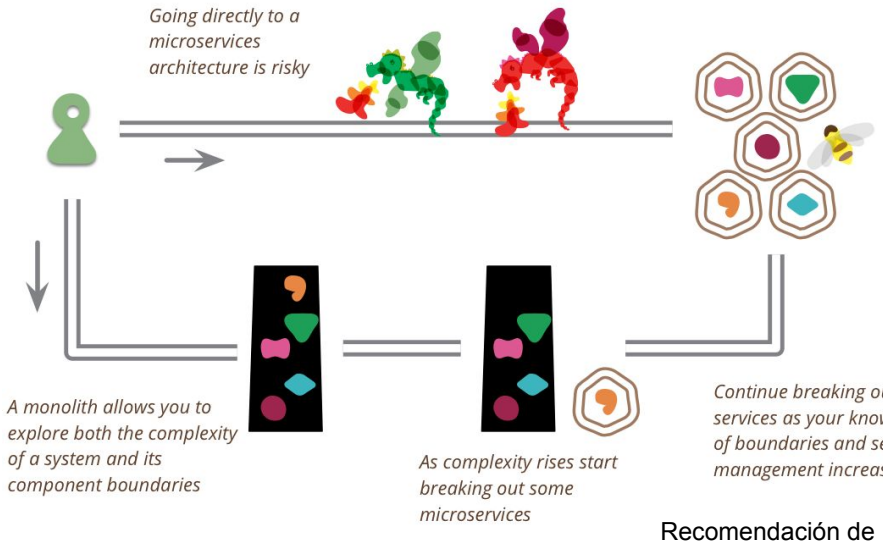
Poll locked. Responses not accepted.

¿Cuáles beneficios tiene una arquitectura de microservicios?

Diversidad de tecnologías
Consistencia eventual
Complejidad operacional
Sistemas distribuidos
Límites de módulos fuertes
Independicia al publicar (deploy)

¿Cuál es la mejor arquitectura de software?

Top



Recomendación de Fowler

Tarea de arquitecturas :)

Ahora hablemos de experticia…

¿Qué tienen en común todas estas personas (sin incluir que son deportistas costarricenses)?

Top

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

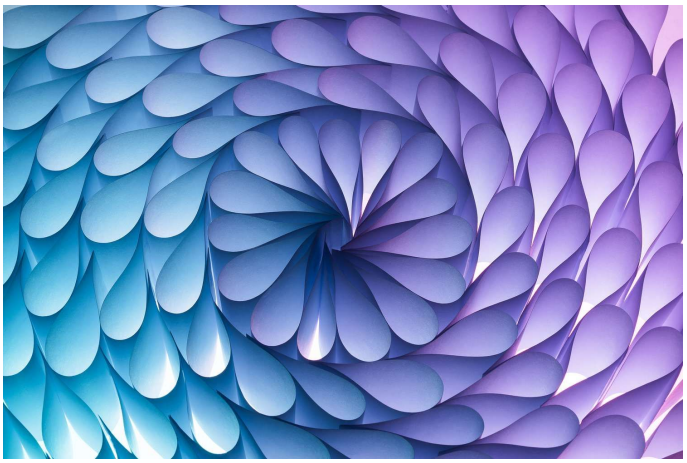
Al programar, uno también gana conocimiento y maestría al resolver problemas

Pongámonos en grupos para resolver un problema :)

¿Cómo es que se recorre un arreglo?

Volvamos a sentarnos…

Los patrones de diseño son soluciones a problemas recurrentes de diseño.



¿Qué es un patrón de software?

Un código de programa escrito previamente que se puede usar para ahorrar tiempo durante el desarrollo de software.

Un conjunto específico de requisitos que deben cumplirse para que un sistema de software funcione correctamente.

Un lenguaje de programación que está optimizado para un tipo particular de desarrollo de software.

Una solución general a un problema recurrente de ingeniería de software, descrita en un formato estandarizado.

Poll locked. Responses not accepted.

¿Qué es un patrón de software?

Un código de programa escrito previamente que se puede usar para ahorrar tiempo durante el desarrollo de software.

Un conjunto específico de requisitos que deben cumplirse para que un sistema de software funcione correctamente.

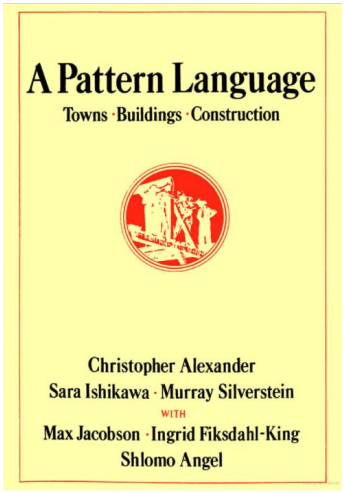
Un lenguaje de programación que está optimizado para un tipo particular de desarrollo de software.

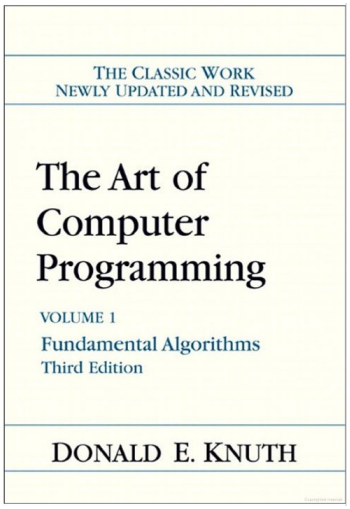
Una solución general a un problema recurrente de ingeniería de software, descrita en un formato estandarizado.

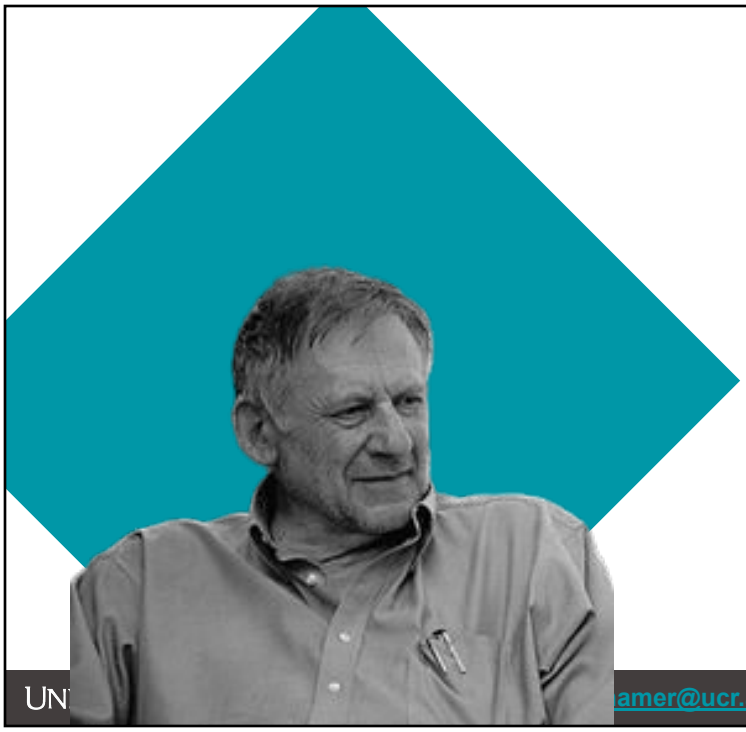
Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

El concepto de un patrón no es un concepto del campo de la ingeniería de software

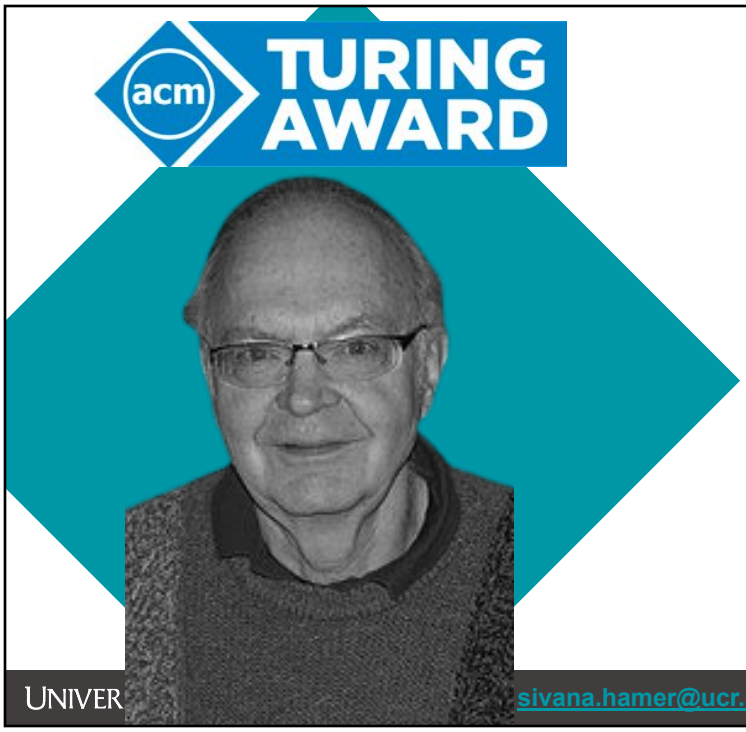






Christopher Alexander

Dr. de Harvard. Sus padres escparon el régimen nazi. Tuvo el primer doctorado de arquitectura en Harvard. Trabajó en Berkeley hasta pensionarse. Su trabajo tuvo un impacto para el campo (e.g., Dijkstra y diseño).



Donald Knuth

Dr. de California Institute of Technology. Su tesis doctoral trabajó en el campo de matemática. Algunos temas que investigó incluyen compiladores, criptografía, AI, y lenguajes de programación.

"*Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.*" Christopher Alexander

Ward Cunningham

Ingeniero de software de Purdue University. Creó el primer wiki y es uno de los autores del manifiesto ágil. Pionero de testing, smalltalk y diseño.

Kent Beck (es toda)

Ingeniero de software graduado de University of Oregon. Creador de extreme programming (XP) y uno de los autores del manifiesto ágil. Pionero de testing, smalltalk y diseño.

Ambos, inspirados por trabajos anteriores, empezaron a trabajar en cinco patrones de diseño para SmallTalk.

Using Pattern Languages for Object-Oriented Programs

Kent Beck, Apple Computer, Inc.
Ward Cunningham, Tektronix, Inc.

Technical Report No. CR-87-43
September 17, 1987

Submitted to the OOPSLA-87 workshop on the
Specification and Design for Object-Oriented Programming.

Abstract

We outline our adaptation of Pattern Language to object-oriented programming. We sumarize a system of five patterns we have successfuly used for designing window-based user interfaces and present in slightly more detail a single pattern drawn from our current effort to record a complete pattern language for object-oriented programs.

The search for an appropriate methodology for object-oriented programming has seen the usual rehash of tired old ideas, but the fact is that OOP is so different that no mere force-fit of structured analysis or entity-relationship methods will provide access to the potential inherent in OOP. In particular, neither of these methods address the user interface design issues that have obviously become of paramount importance. In addition, while E-R seems to be "object-oriented" it is not suited to the dynamic nature of objects as in Smalltalk and encourages the use of a global perspective while designing, a sure lose in object-oriented programming.

http://c2.com/doc/oopsla87.html

Bruce Anderson

Profesor en University of Essex. Investigo temas de arquitectura y diseño de software.

Erich Gamma

Dr. de University of Zurich. Su tesis doctoral trabajó en patrones de diseño. También trabajó en otros temas como testing. Actualmente trabaja para Microsoft.

Anderson dio una charla del tema y al Dr. Gamma le interesó

Information | Discussion (0) | Files | Holdings

Conference

Conference title	**2nd International Conference on the Technology of Object-oriented Languages and Systems**
Related conference title(s)	TOOLS 2
Date(s), location	25 - 29 Jun 1990, Paris, France
Editor(s)	Bézivin, Jean (ed.) ; Meyer, Bertrand (ed.) ; Nerson, Jean Marc (ed.)
Imprint	Paris : Angkor, 1990 - 702 p.
ISBN	2878920007 9782878920000
Subject category	Computing and Computers
Free keywords	analysis ; design ; distribution ; expert systems ; panel ; toolkit ; tutorial

Richard Helm

Dr. de University of Melbourne. Trabajó dentro de IBM Consulting Group.

Dr. Gamma y Dr. Helm se conocieron en…

Conference
SPLASH
SPLASH: Systems, Programming, and Applications

Search within SPLASH

Home > Conferences > SPLASH > Proceedings > OOPSLA '91

OOPSLA '91: Conference proceedings on Object-oriented programming systems, languages, and applications

Proceeding

1991 Proceeding

Editor: Andreas Paepcke

Publisher: Association for Computing Machinery, New York, NY, United States

Conference: OOPSLA 91: Object Orientated Programming Systems and Applications • Phoenix Arizona USA • October 6 - 11, 1991

ISBN: 978-0-201-55417-5

Published: 01 November 1991

Sponsors: SIGPLAN

Get Alerts for this Conference

Save to Binder

Export Citation

Ralph Johnson

Dr. de Cornell. Trabajó como profesor en el departamento de computación en University of Illinois at Urbana-Champaign. Además investigó como reusabilidad y arquitecturas.

John Vlissides

Dr. de Stanford en ingeniería eléctrica. Trabajó en IBM y Standford. Investigó otros temas como reusabilidad y generación de código.

Los otros dos se unieron luego y ellos son conocidos como “Gang of Four” (GoF)

Ralph Johnson Erich Gamma Richard Helm John Vlissides

Publicaron un artículo académico al respecto.

Design Patterns: Abstraction and Reuse of Object-Oriented Design

Erich Gamma[1*], Richard Helm[2], Ralph Johnson[3], John Vlissides[2]

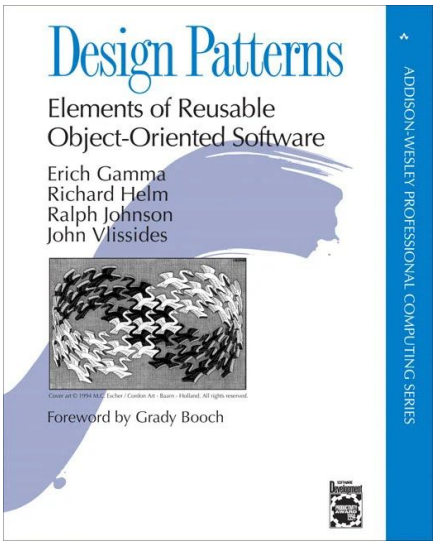
[1] Taligent, Inc.
10725 N. De Anza Blvd., Cupertino, CA 95014-2000 USA

[2] I.B.M. Thomas J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598 USA

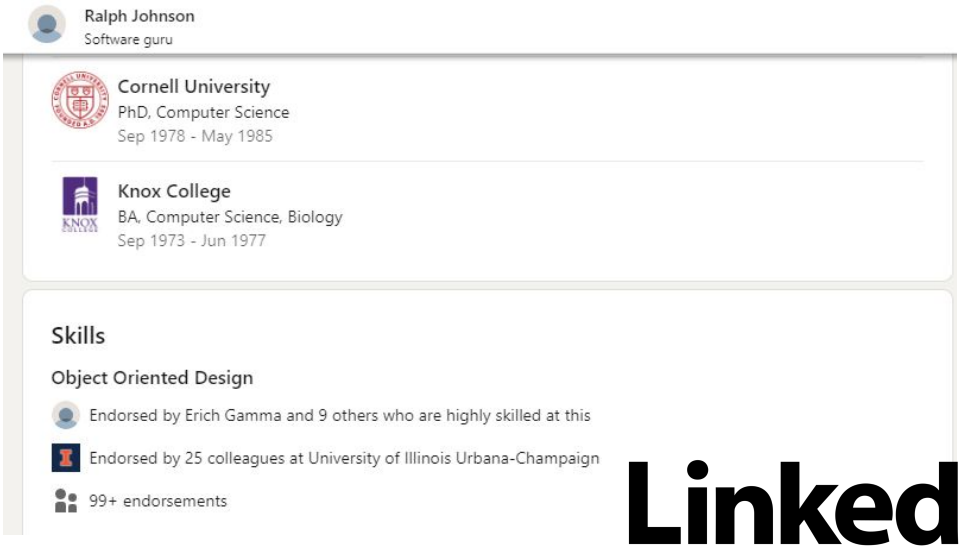
[3] Department of Computer Science
University of Illinois at Urbana-Champaign
1034 W. Springfield Ave., Urbana, IL 61801 USA

Abstract. We propose **design patterns** as a new mechanism for expressing object-oriented design experience. Design patterns identify, name, and abstract common themes in object-oriented design. They capture the intent behind a design by identifying objects, their collaborations, and the distribution of responsibilities. Design patterns play many roles in the object-oriented development process: they provide a common vocabulary for design, they reduce system complexity by naming and defining abstractions, they constitute a base of experience for building reusable software, and they act as building blocks from which more complex designs can be built. Design patterns can be considered reusable micro-architectures that contribute to an overall system architecture. We describe how to express and organize design patterns and introduce a catalog of design patterns. We also describe our experience in applying design patterns to the design of object-oriented systems.

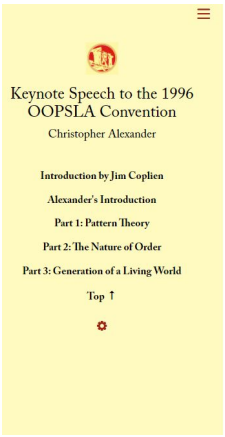
Son los autores de este libro que popularizó el tema.



Quite the times we live in…



Incluso el Dr. Alexander dio una charla en OOPSLA'96 al respecto del tema.



THE ORIGINS OF PATTERN THEORY,
THE FUTURE OF THE THEORY, AND THE GENERATION OF A LIVING WORLD

Keynote Speech to the 1996 OOPSLA Convention
By Christopher Alexander

The following presentation was recorded live in San Jose, California, October of 1996, at The 1996 ACM Conference on Object-Oriented Programs, Systems, Languages and Applications (OOPSLA).

Introduction by Jim Coplien

Once in a great while, a great idea makes it across the boundary of one discipline to take root in another. The adoption of Christopher Alexander's patterns by the software community is one such event. Alexander both commands respect and inspires controversy in his own discipline; he is the author of several books with long-running publication records, the first recipient of the AIA Gold Medal for Research, a member of the Swedish Royal Academy since 1980, a member of the American Academy of Arts and Sciences, recipient of dozens of awards and honors including the Best Building in Japan award in 1985, and the American Association of Collegiate Schools of Architecture Distinguished Professor Award. It is odd that his ideas should have found a home in software, a discipline that deals not with timbers and tiles but with pure thought stuff, and with ephemeral and weightless products called programs. The software community embraced the pattern vision for its relevance to problems that had long plagued software design in general and object-oriented design in particular. Focusing on objects had caused us to loose the system perspective. Preoccupation with design method had caused us to loose the human perspective. The curious parallels between Alexander's world of buildings and our world of software construction helped the ideas to take root and thrive in grass-roots programming communities world-wide. The pattern discipline has become one of the most widely applied and important ideas of the past decade in software architecture and design.

https://www.patternlanguage.com/archive/ieee.html

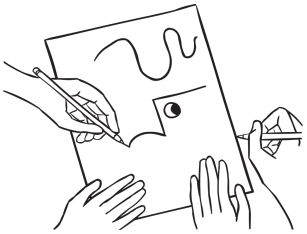
En el libro proponen 22 patrones de diseño

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method	Adapter	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Flyweight (195) Observer State Strategy Visitor

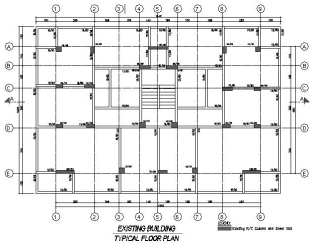
Gamma et al.

Son tres tipos de clasificación de propósito

Creacionales
Encargados del proceso de creación

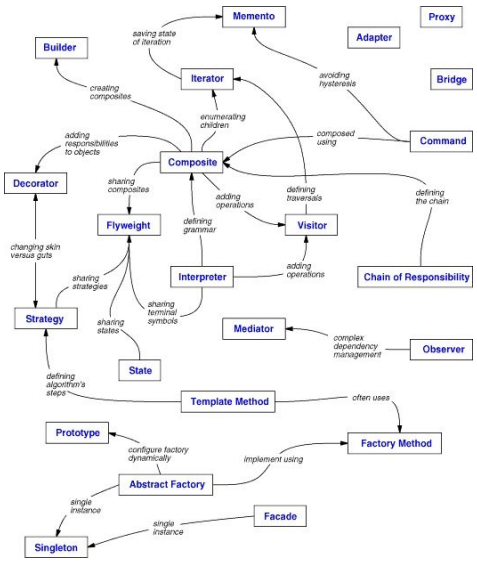


Estructurales
Describen cómo se componen

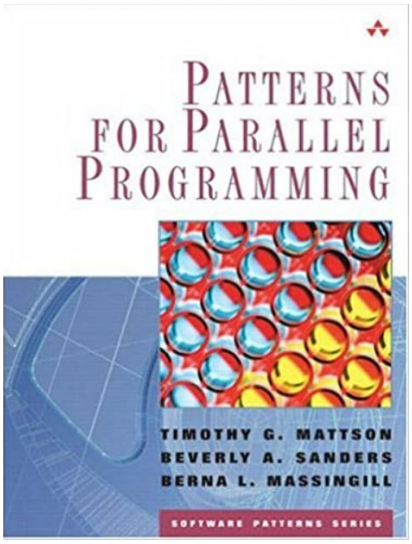


Comportamiento
Caracterizan como interactúan y distribuyen responsabilidades

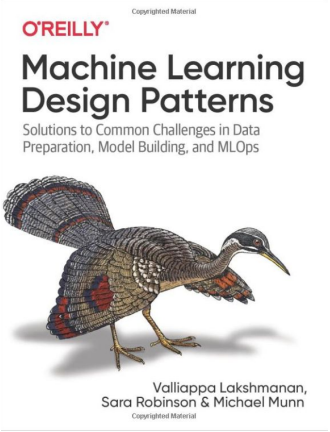
Los GoF design patterns son…

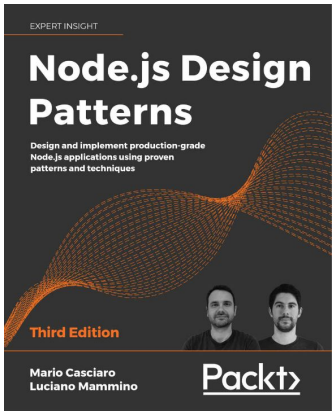


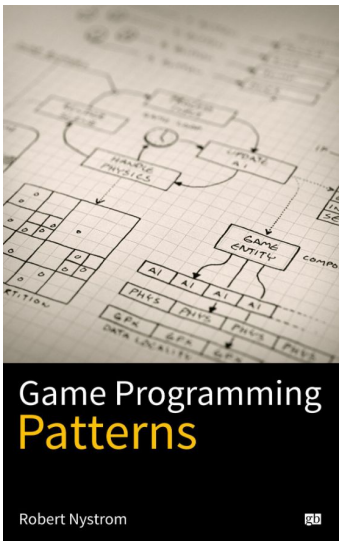
También veremos patrones de programación paralela



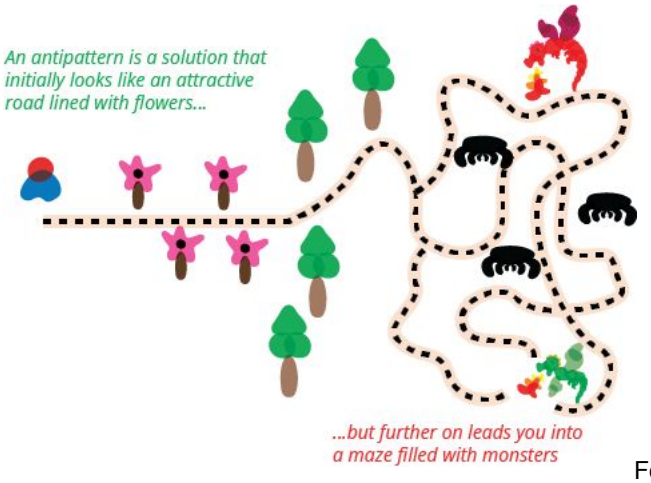
Aunque lo que vamos a ver solo va a ser una pincelada…







Un antipatrón es como parece una buena solución inicial pero no lo es.



Fowler

También existen los olores de código que son implementaciones que pueden indicar problemas en el código

Poll locked. Responses not accepted.

¿Es una arquitectura lo mismo que un patrón de software?

Si

No

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

Poll locked. Responses not accepted.

¿Es una arquitectura lo mismo que un patrón de software?

Si

No

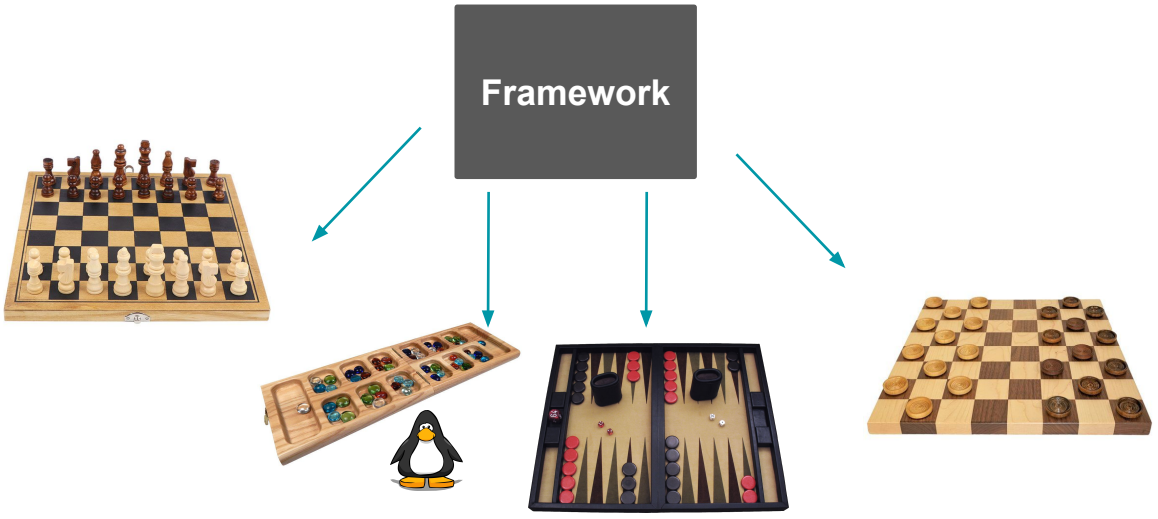
Powered by Poll Everywhere

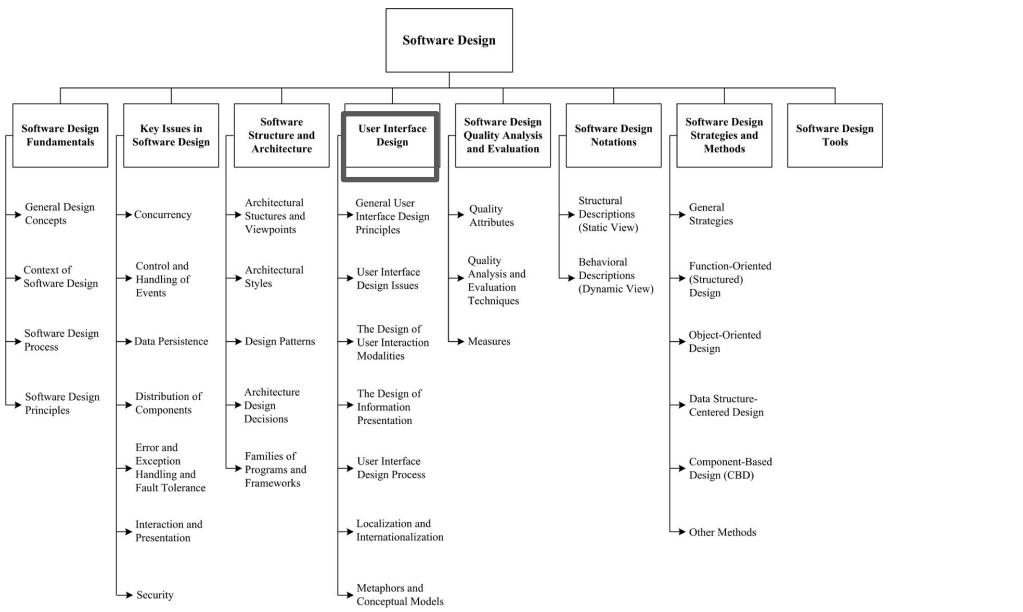
Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

La arquitectura de un software es más alto nivel en el diseño de un sistema que un patrón de diseño

Un framework es un diseño reusable de un sistema total o parcial.

Un ejemplo de un framework dentro de un juego…





INICIO > INTERACCIÓN HUMANO-COMPUTADOR

Interacción Humano-Computador

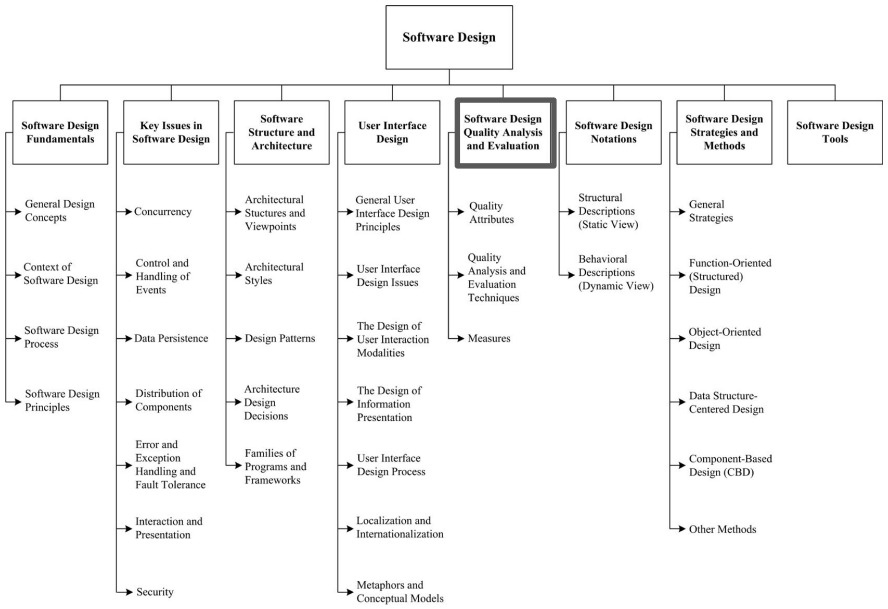
Atributos

Sigla:	CI-0135
Créditos:	4
Horas:	5
Requisitos:	Programación 2
Clasificación:	Curso propio
Énfasis y ciclo:	Ingeniería de Software 3.I

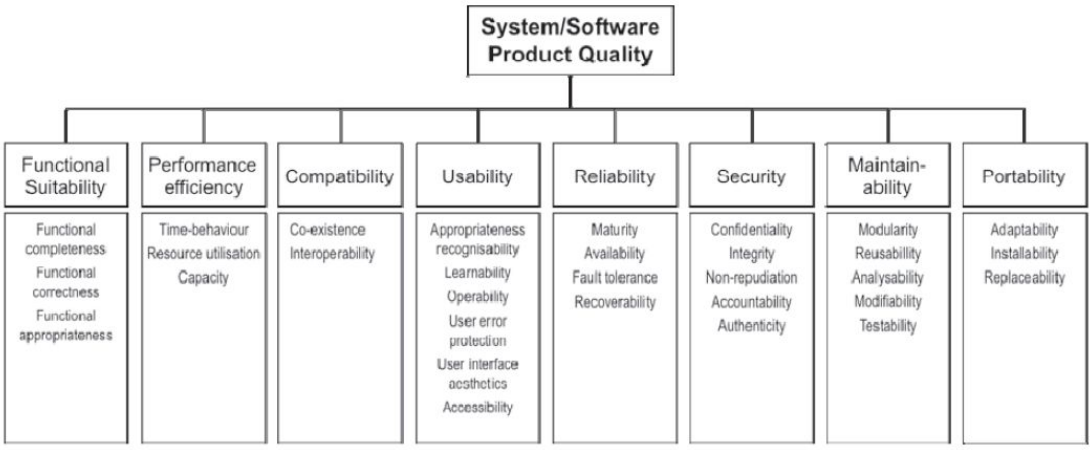
Descripción:
El estudio de la Interacción Humano-Computador (IHC) es fundamental para diseñar y crear aplicaciones que sean útiles y fáciles de usar para los usuarios a quienes van dirigidas. En este curso se introducen los principios básicos de diseño de aplicaciones, así como las técnicas fundamentales de evaluación de la interacción. El curso contempla la teoría básica de diseño y evaluación de usabilidad. También se desarrollan prototipos para poner en práctica los conceptos teóricos aprendidos en el curso.

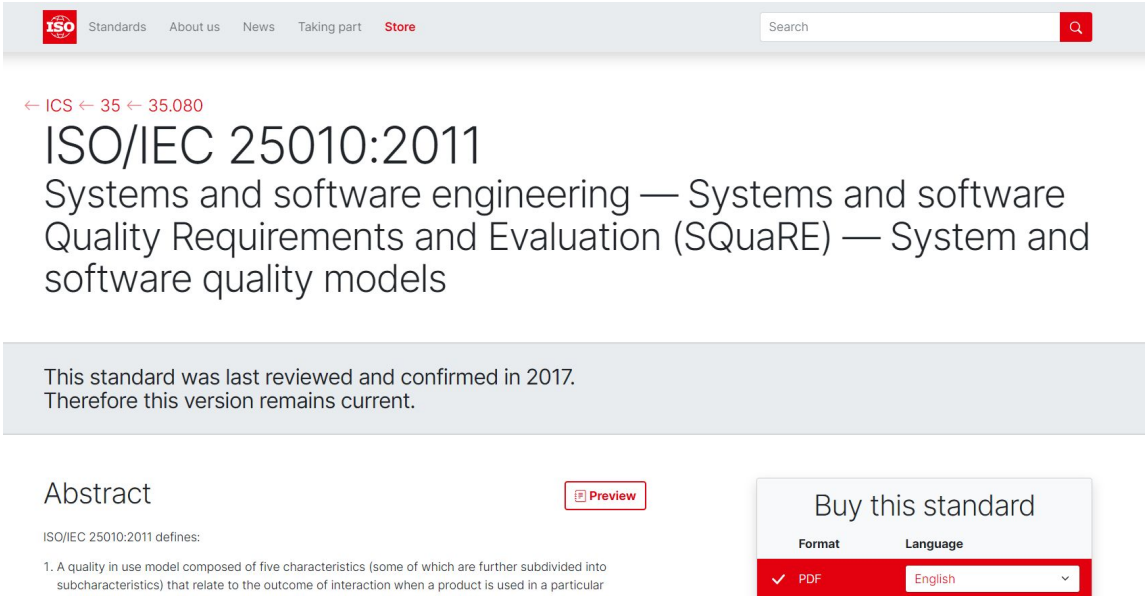
Objetivo general:
El objetivo general del curso es que los estudiantes sean capaces de aplicar técnicas de interacción y usabilidad para diseñar y evaluar interfaces de usuario, mediante su uso en interfaces diseñadas por ellos y en la evaluación de interfaces creadas por terceros.

Objetivos específicos:

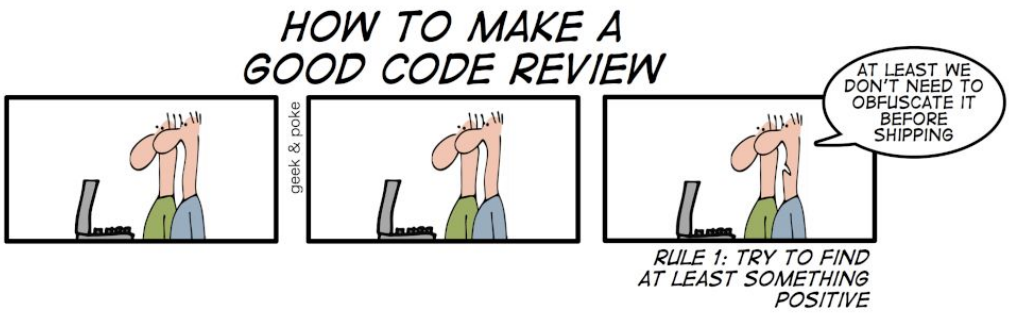


¿Cómo determinan que un software es de buena cualidad?

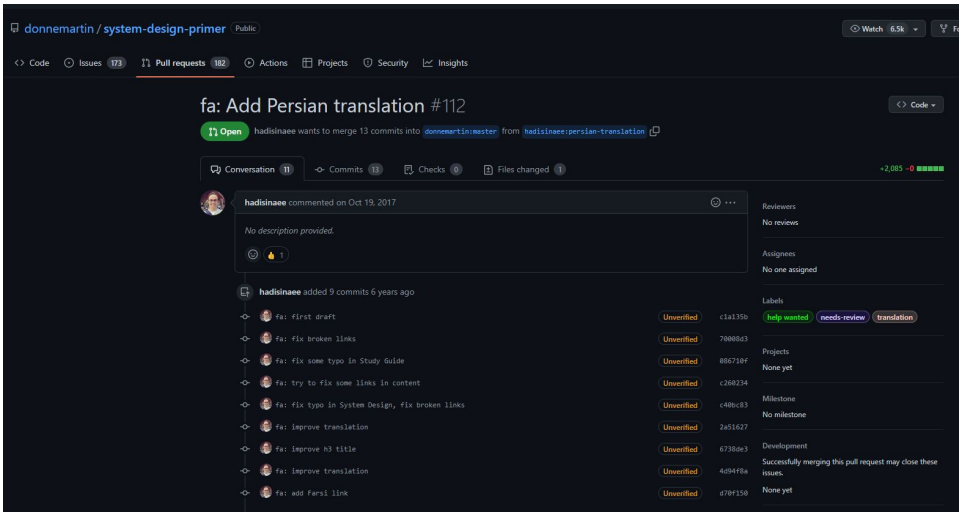

System/Software Product Quality
Functional Suitability
Functional completeness
Functional correctness
Functional appropriateness
Performance efficiency
Time-behaviour
Resource utilisation
Capacity
Compatibility
Co-existence
Interoperability
Usability
Appropriateness recognisability
Learnability
Operability
User error protection
User interface aesthetics
Accessibility
Reliability
Maturity
Availability
Fault tolerance
Recoverability
Security
Confidentiality
Integrity
Non-repudiation
Accountability
Authenticity
Maintain-ability
Modularity
Reusability
Analysability
Modifiability
Testability
Portability
Adaptability
Installability
Replaceability


Standards
About us
News
Taking part
Store
Search
← ICS ← 35 ← 35.080
ISO/IEC 25010:2011
Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models
This standard was last reviewed and confirmed in 2017.
Therefore this version remains current.
Abstract
Preview
ISO/IEC 25010:2011 defines:
1. A quality in use model composed of five characteristics (some of which are further subdivided into subcharacteristics) that relate to the outcome of interaction when a product is used in a particular
Buy this standard
Format
Language
PDF
English

Una manera de evaluar la calidad es por medio de reviews



En GitHub se hacen reviews de código por medio de pull requests



Siempre es bueno tener una lista de chequeo o el definition of done…

Otra manera es utilizando técnicas de análisis estático

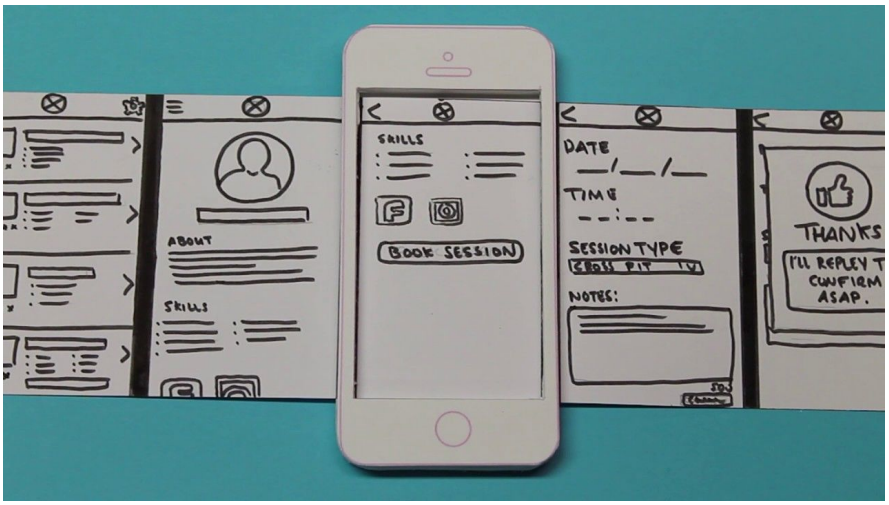
sonarqube

CODACY

UNIVERSITY OF MARYLAND 18 56

FindBugs™
because it's easy

Otras técnicas incluyen simulación y prototipado



Las mediciones se pueden utilizar para evaluar aspectos de la calidad del software

¿Cómo medirían la calidad de un software?

Top

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Goal	Attribute	Metric
Requirement quality	Ambiguity	Number of ambiguous modifiers (e.g., many, large, human-friendly)
	Completeness	Number of TBA, TBD
	Understandability	Number of sections/subsections
	Volatility	Number of changes per requirement
		Time (by activity) when change is requested
	Traceability	Number of requirements not traceable to design/code
	Model clarity	Number of UML models
		Number of descriptive pages per model
		Number of UML errors
Design quality	Architectural integrity	Existence of architectural model
	Component completeness	Number of components that trace to architectural model
		Complexity of procedural design
	Interface complexity	Average number of pick to get to a typical function or content
		Layout appropriateness
	Patterns	Number of patterns used
Code quality	Complexity	Cyclomatic complexity
	Maintainability	Design factors (Chapter 8)
	Understandability	Percent internal comments
		Variable naming conventions
	Reusability	Percent reused components
	Documentation	Readability index
QC effectiveness	Resource allocation	Staff hour percentage per activity
	Completion rate	Actual vs. budgeted completion time
	Review effectiveness	See review metrics (Chapter 14)
	Testing effectiveness	Number of errors found and criticality
		Effort required to correct an error
		Origin of error

Pressman

When poll is active, respond at **pollev.com/sivanahamer104**

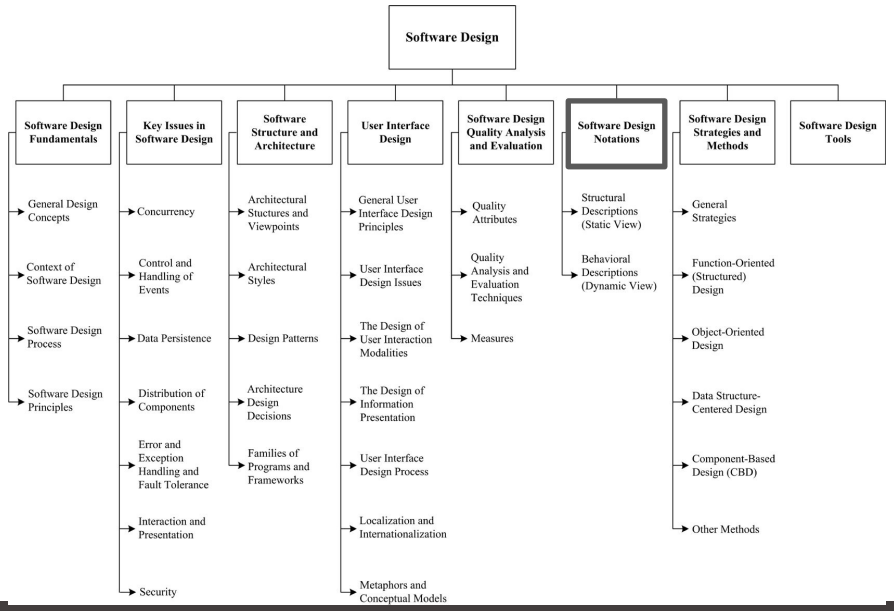
¿Han verificado ustedes la calidad de software con un método o una métrica formal?

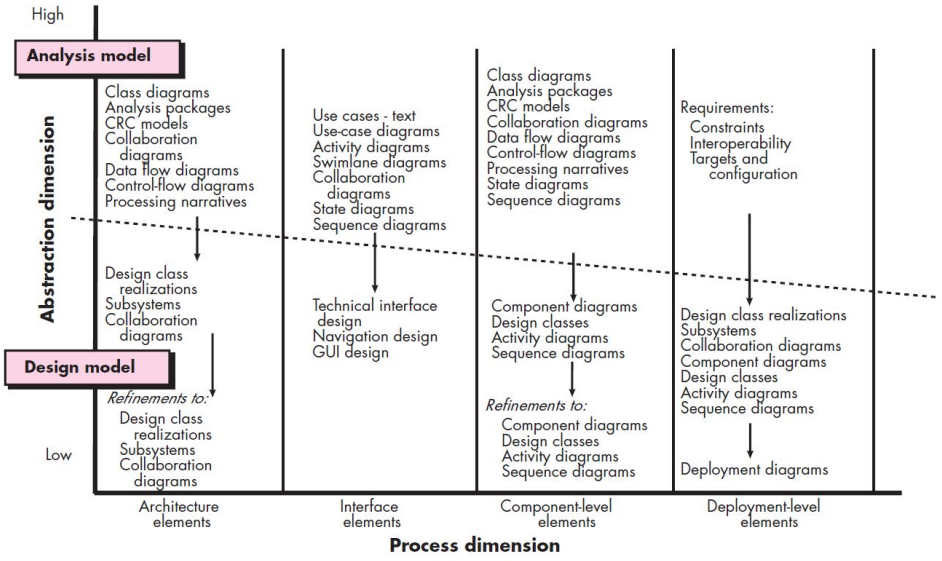
Si

No

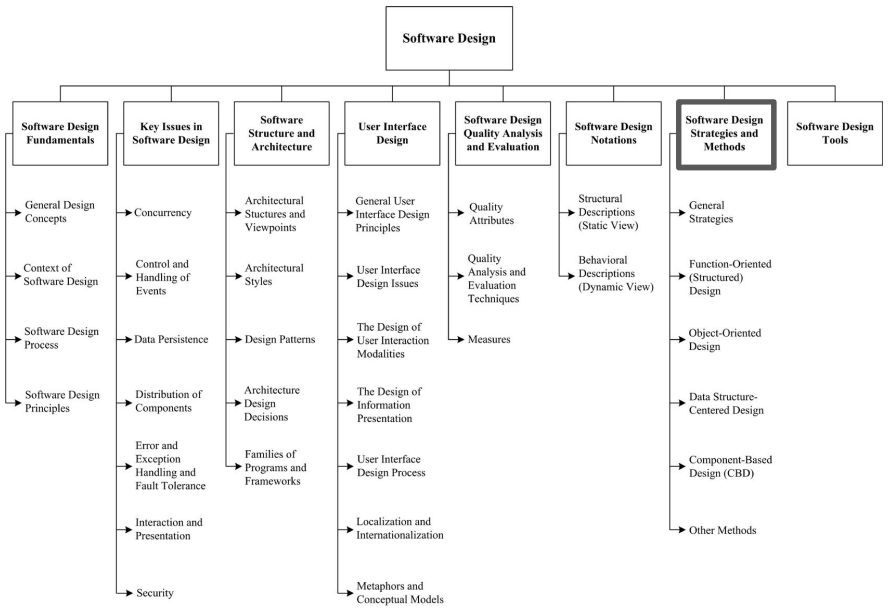
Powered by Poll Everywhere

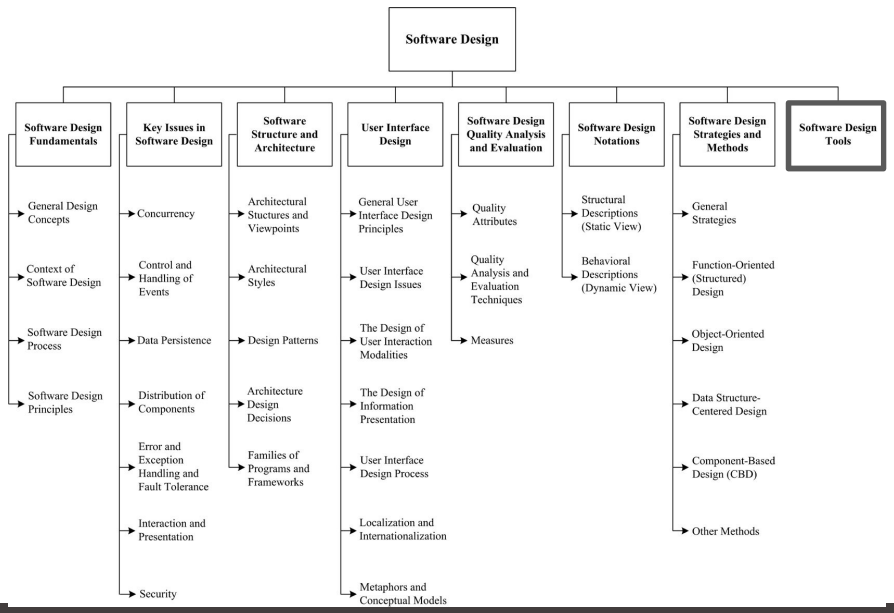
Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**




High
Analysis model
Abstraction dimension
Class diagrams
Analysis packages
CRC models
Collaboration diagrams
Data flow diagrams
Control-flow diagrams
Processing narratives
Use cases - text
Use-case diagrams
Activity diagrams
Swimlane diagrams
Collaboration diagrams
State diagrams
Sequence diagrams
Class diagrams
Analysis packages
CRC models
Collaboration diagrams
Data flow diagrams
Control-flow diagrams
Processing narratives
State diagrams
Sequence diagrams
Requirements:
Constraints
Interoperability
Targets and configuration
Design class realizations
Subsystems
Collaboration diagrams
Technical interface design
Navigation design
GUI design
Component diagrams
Design classes
Activity diagrams
Sequence diagrams
Design class realizations
Subsystems
Collaboration diagrams
Component diagrams
Design classes
Activity diagrams
Sequence diagrams
Design model
Refinements to:
Design class realizations
Subsystems
Collaboration diagrams
Refinements to:
Component diagrams
Design classes
Activity diagrams
Sequence diagrams
Deployment diagrams
Low
Architecture elements
Interface elements
Component-level elements
Deployment-level elements
Process dimension


COMING
SOON


Software Design
Software Design Fundamentals
General Design Concepts
Context of Software Design
Software Design Process
Software Design Principles
Key Issues in Software Design
Concurrency
Control and Handling of Events
Data Persistence
Distribution of Components
Error and Exception Handling and Fault Tolerance
Interaction and Presentation
Security
Software Structure and Architecture
Architectural Stuctures and Viewpoints
Architectural Styles
Design Patterns
Architecture Design Decisions
Families of Programs and Frameworks
User Interface Design
General User Interface Design Principles
User Interface Design Issues
The Design of User Interaction Modalities
The Design of Information Presentation
User Interface Design Process
Localization and Internationalization
Metaphors and Conceptual Models
Software Design Quality Analysis and Evaluation
Quality Attributes
Quality Analysis and Evaluation Techniques
Measures
Software Design Notations
Structural Descriptions (Static View)
Behavioral Descriptions (Dynamic View)
Software Design Strategies and Methods
General Strategies
Function-Oriented (Structured) Design
Object-Oriented Design
Data Structure-Centered Design
Component-Based Design (CBD)
Other Methods
Software Design Tools


Software Design
Software Design Fundamentals
General Design Concepts
Context of Software Design
Software Design Process
Software Design Principles
Key Issues in Software Design
Concurrency
Control and Handling of Events
Data Persistence
Distribution of Components
Error and Exception Handling and Fault Tolerance
Interaction and Presentation
Security
Software Structure and Architecture
Architectural Stuctures and Viewpoints
Architectural Styles
Design Patterns
Architecture Design Decisions
Families of Programs and Frameworks
User Interface Design
General User Interface Design Principles
User Interface Design Issues
The Design of User Interaction Modalities
The Design of Information Presentation
User Interface Design Process
Localization and Internationalization
Metaphors and Conceptual Models
Software Design Quality Analysis and Evaluation
Quality Attributes
Quality Analysis and Evaluation Techniques
Measures
Software Design Notations
Structural Descriptions (Static View)
Behavioral Descriptions (Dynamic View)
Software Design Strategies and Methods
General Strategies
Function-Oriented (Structured) Design
Object-Oriented Design
Data Structure-Centered Design
Component-Based Design (CBD)
Other Methods
Software Design Tools

Algunos ejemplos de herramientas…

Lucidchart

Referencias

- Pressman, R. S. (2010). Software engineering: a practitioner's approach. Palgrave macmillan.
- Lakshmanan, V., Robinson, S., & Munn, M. (2020). Machine learning design patterns. O'Reilly Media.
- Nystrom, R. (2014). Game programming patterns. Genever Benning.
- Casciaro, M., & Mammino, L. (2016). Node. js Design Patterns. Packt Publishing Ltd.
- Fowler, M. (2019). Software architecture guide. Recovered from: https://martinfowler.com/architecture/
- Alexander, C. (1977). A pattern language: towns, buildings, construction. Oxford university press.

Referencias

- Knuth, D. E. (1997). The art of computer programming. Pearson Education.
- History of Patterns (2011). Recuperado de: http://wiki.c2.com/?HistoryOfPatterns
- Bourque, P., & Fairley, R. E. (2014). Guide to the Software Engineering Body of Knowledge (SWEBOK (R)) Version 3.0.
- Shvets, A. (2019). Dive into Design Patterns.
- Schmidt, D. (s.f.). Overview of Patterns: Introduction. Recuperado de: https://www.dre.vanderbilt.edu/~schmidt/cs282/PDFs/3-PatternOverview.pdf
- Hadeli, M. (202). Awesome Software Architecture. Recuperado de: https://awesome-architecture.com/

Referencias

- Varios. (2022). The System Design Primer. Recuperado de: https://github.com/donnemartin/system-design-primer
- Gamma, E., Johnson, R., Helm, R., Johnson, R. E., & Vlissides, J. (1995). Design patterns: elements of reusable object-oriented software. Pearson Deutschland GmbH.
- Freeman, E., Freeman, E., Bates, B., & Sierra, K. (2004). Head First Design Patterns.
- Vijini Mallawaarachchi. (2017). 10 Common Software Architectural Patterns in a nutshell. Recuperado de: https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013

Referencias

- Richards, M. (2015). Software Architecture Patterns.
- Martin, R. C. (2017). Clean architecture.
- Giordani, L. (2022). Clean architecture in Python.
- Zimmerman, J. (2020). Don't Mock Your Domain. Recuperado de: https://programmingideaswithjake.wordpress.com/2020/07/27/dont-mock-your-domain/
- Johnson, R. E. (1997). Frameworks=(components+ patterns). Communications of the ACM, 40(10), 39-42.
- Stanford. (2012). How We've Scaled Dropbox. Recuperado de: https://www.youtube.com/watch?v=PE4gwstWhmc

Referencias

- Educative.io (s.f.). System design. Recuperado de: https://www.educative.io/courses/grokking-modern-system-design-interview-for-engineers-managers?affiliate_id=5073518643380224
- Xu, A., & Lam, S. (2022). System Design Interview: An Insider's Guide. Byte Code LLC.
- Wahome. (2020). Concurrency is not Parallelism. Recuperado de: https://kwahome.medium.com/concurrency-is-not-parallelism-a5451d1cde8d
- Orenstein, G., Doherty, C., White, K., & Camina, S. (2015). Building Real-Time Data Pipelines. O'Reilly Media, Incorporated.
- Maqsood, T., Finegan, A. D., & Walker, D. H. (2003, December). A soft approach to solving hard problems in construction project management. In 2nd International Conference on Construction in the 21st Century (CITC-II), Hong Kong (pp. 10-12).

Referencias

- Petersen, K. (2011). Is lean agile and agile lean?: a comparison between two software development paradigms. In Modern software engineering concepts and practices: Advanced approaches (pp. 19-46). IGI Global.