Introducción:

Software Development Methodology: Splitting software into different stages with the purpose of more effective planning and management.

Stages:

- Requerimientos
- Desarollo
- Pruebas
- Lanzamiento

Metodologías de Software:

- Waterfall/Traditional: For a small application with clear-cut instructions, that needs about 100 hours to implement. Obsolete when a lot of changing and tweaking is needed.
- Scrum: Opposite of Waterfall. Agile iterative strategy for big projects. Top priority tasks are implemented first. Sprints of 2 to 4 weeks of duration. Can change product requirements.
- Kanban: Agile methodology. Milestones rather than velocity. Complex projects with rapidly emerging changes. Cheap
- Scrumban
- Extreme Programming: Fast. More specialists.

Roles en Desarrollo de Software:

- Producto o Manejo de Proyecto
- Equipo de Desarollo
- Quality Assurance
- DevOps / Integración Continua

Arquitectura:

Definición: El proceso de definir la estructura de un sistema para que cumpla con los requerimientos técnicos y operacionales.

- **Arquitectura por capas:** Un ejemplo de arquitectura por capas es Gmail. Este software tiene una capa de presentación que comunica al usuario la información que desea. Esta capa es la que ves cuando ingresas para revisar tu mail. También tiene una capa de tipo “business” donde ocurre todo el procesamiento del sistema, incluyendo las interacciones que el usuario hace con el sistema. En esta capa existe la lógica que manda correos. Finalmente, existe una capa de base de datos donde están guardados todos los mails de los usuarios y otros tipos de mensaje.

- **Arquitectura por eventos:** Un ejemplo de una aplicación de arquitectura por eventos lo podemos ver en el ámbito público, en los **semáforos**. En algunos lugares, sobre intersecciones con semáforos, se instalan sistemas con cámaras que detectan cuando un auto se cruza el alto. Si ocurre esto, el sistema registra la ocurrencia de un evento que puede incluir el registro de una imágen, localización y tiempo. Después el sistema también podría generar un reporte de la infracción cometida.

- **Microkernel:** Un ejemplo relativamente popular de la arquitectura de microkernel es el IDE de Eclipse. La base de Eclipse es un editor de código, sin embargo tiene una vasta selección de plug-ins distintos para diferentes funcionalidades. El Eclipse inicial que el usuario obtiene cuando solamente baja el IDE es el “Core System”. Un ejemplo de un “Plug-In Component” sería Maven, un add-on para Eclipse que facilita el desarrollo de aplicaciones y proyectos en Java.

- **Microservicios:** Un ejemplo muy popular de la arquitectura de microservicios es Amazon. A principios de los 2000, Amazon no tenía arquitectura de microservicios y esto causó que fuera muy complicado escalar el tamaño de la aplicación y agregar funcionalidades nuevas porque los sistemas existentes eran muy interdependientes. Para solucionar este problema, los ingenieros de Amazon rediseñaron la arquitectura de Amazon a una de microservicios, en la cual dividieron el sistema original en pequeñas unidades que estaban enfocadas en propósitos específicos. Por ejemplo, la función “Buy” fue empaquetada dentro de un microservicio. Otro ejemplo es que el cálculo de impuesto de la compra de algo se empaquetó en un microservicio.

- **Space Based Architecture:**

	Layered	Event-driven	Microkernel	Microservices	Space-based
Overall Agility	↓	↑	↑	↑	↑
Deployment	↓	↑	↑	↑	↑
Testability	↑	↓	↑	↑	↓
Performance	↓	↑	↑	↓	↑
Scalability	↓	↑	↓	↑	↑
Development	↑	↓	↓	↑	↓

¿Qué debemos considerar?

- Desarollo
- Deployment/Despliegue
- Operaciones
- Mantenimiento

Administración de proyectos:

Ir de la idea al lanzamiento.

Administrar:
- Roles, responsabilidades y metodologías.
- Requerimientos, commitments y entregables.

Roles y responsabilidades:
- Designer
- Backend
- Devops
- Frontend
- Quality Assurance
- Product Manager
- Business Owner

¿Que es una metodología?
Una manera de llevar el desarrollo de funcionalidades de manera que nos lleve a entregar valor a un cliente y/o usuario.

Metodologías:
- Cascada: No te puedes regresar del siguiente proceso: Definición, desarrollo, pruebas, lanzamiento.
- Por prototipos: visual. Aprovechar el prototipo para definir ideas. “Construir una maqueta.”
- Agile: En cualquier momento puedes hacer cambios. Muy exitosa. Iteración y rapidez.
- Rapid:
- Dinámica: Basado en factibilidad y modelo de negocios, definimos las funcionalidades e implementamos.
- Espiral: Funcionalidades de lo más chico a lo más grande.
- Extreme Programming: Tan pronto tengas idea de lo que vas a hacer, hazlo. Es como un hackathon.
- Feature Driven: Construyó conforme vayan llegando las ideas de funcionalidades. No hay lógica tan clara en todo el proyecto. Luego falla.

Los requerimientos, roadmap y features definen lo tiempos y entregables del proyecto. Es decir, como le digo a alguien que hacer en el proyecto.

Feature: “A distinguishing characteristic of a software item (performance, portability or functionality).”

Examples:

- Add a student to a seminar waiting list.
- Calculate fee for a parking pass.
- Calculate the average mark on a transcript.
- Display the name and address of a student on a transcript.

Definición de Requerimientos:

- Desde 1998, el template de IEEE se ha utilizado mucho.
- 4 secciones:
    - Introducción: Scope, definitions, purpose, overview.
    - Descripcion Overall: Functionality, characteristics, constraining elements.
    - Requerimientos específicos:
        - Functions
        - Interfaces
        - Performance and data requirements
        - Design constraints
        - Security
        - Maintenance
        - Features and system attributes
        - Priorities
        - Release plans
    - Información de Soporte: apéndices.

Los requerimientos deben ser:

- Correctos: método de análisis que asegura que el software cumple los requerimientos identificados.
- No ambiguos: Solo tienen una interpretación.
- Completos: Representan todos los requerimientos de funcionalidad, desempeño, límites de diseño, atributos e interfaces.
- Consistentes: No contradicen ninguna otra documentación.
- Rankeados de acuerdo a importancia y/o estabilidad: Employ a method to appropriately rank requirements.
- Verificables: Use measurable elements and defined terminology to avoid ambiguity.
- Modificables: La estructura de organización del SRS debe evitar redundancia y permitir adaptación fácil.
- Rastreables: Habilidad de rastrear de regreso el origen de el desarollo.

Preguntas útiles:

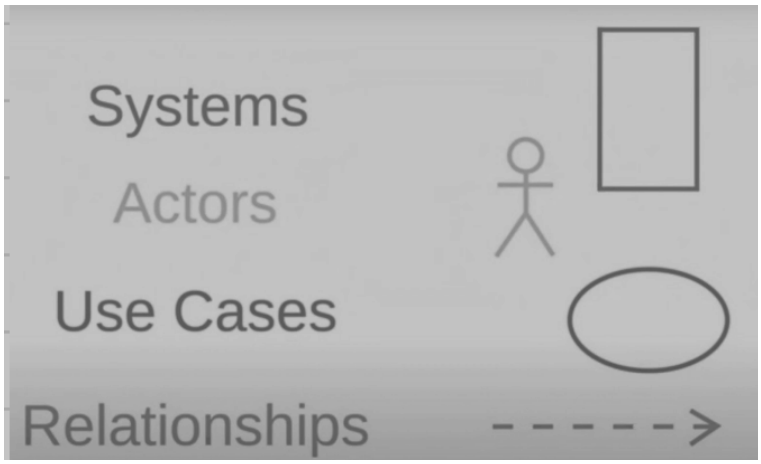
- ¿Qué problemas enfrentan hoy las personas que el software no soluciona?
- ¿Qué problemas quieres que solucione el software?
- ¿Hay procesos manuales que te gustaría automatizar?

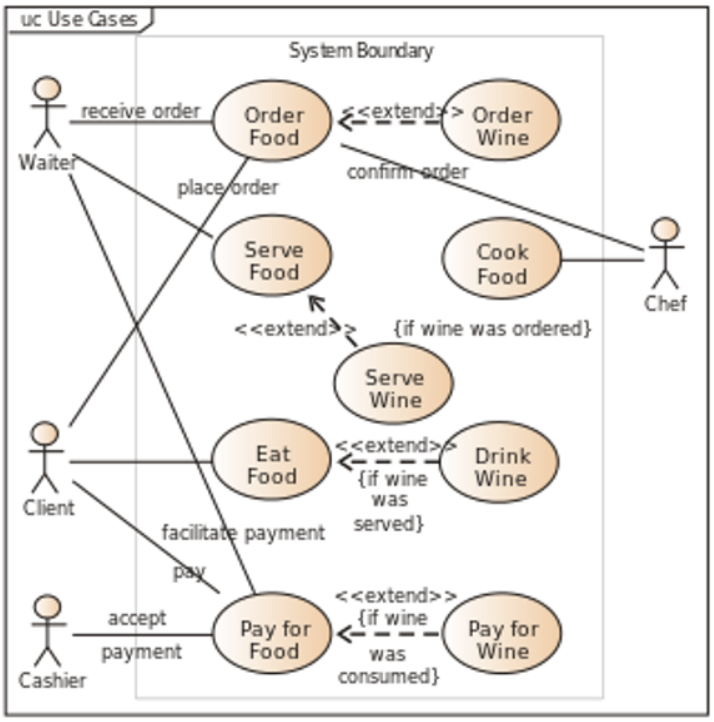
- ¿Qué datos quieres recolectar?

- ¿Qué haría que el software sea exitoso?
- ¿Cómo se vería la funcionalidad “xyz”?
- ¿Quién usará esa funcionalidad?
- ¿Quién tendrá acceso a la funcionalidad?

Casos de uso: Son los pasos que sigue el usuario para lograr un objetivo.

Diagrama UML: Standardized modeling language, universal.





Actors: Primary (customer) and Secondary (reactionary)
Casos de uso: verbos que representan una acción.
Relationship: association, include, extend, generalization

Métricas y estimaciones del software:

Seguridad en el desarrollo de software:

Software seguro es software que continúa funcionando aunque esté bajo ataque.

¿Qué medir?
C.IA. Triad: Confidentiality, Integrity and Availability.

A.A.A. Authentication, Authorization and Accountability.

Confidencialidad: Protecciones contra revelar datos sensibles y no autorizados de individuos. Se debe considerar que los datos sean seguros cuando se envían, están almacenados y cuando están en proceso. Normalmente se usan procesos como encriptación.

Integridad: Debe asegurarse de que los datos son correctos y confiables, es decir que los datos son modificados por las personas adecuadas y que sean consistentes mientras son manejados. Para cumplir con estos requerimientos se usa Hashing, Firmas digitales, protocolos y listas de seguridad.

Disponibilidad: Se asegura de que los datos no se destruyan cuando no deben. Se usan Service Level Agreements, Tiempos máximos de tolerancia y Tiempos de rescate.

Autenticación: Valida la identidad de las personas a través de tokens, tarjetas smart, biometría, autenticación de dos pasos, etc.

Autorización: Los permisos adecuados son asignados a las personas correctas por medio de roles y reglas.

Rendición de cuentas: Construir los mecanismos adecuados para reportar las acciones de los usuarios, normalmente se realiza a través de logs.

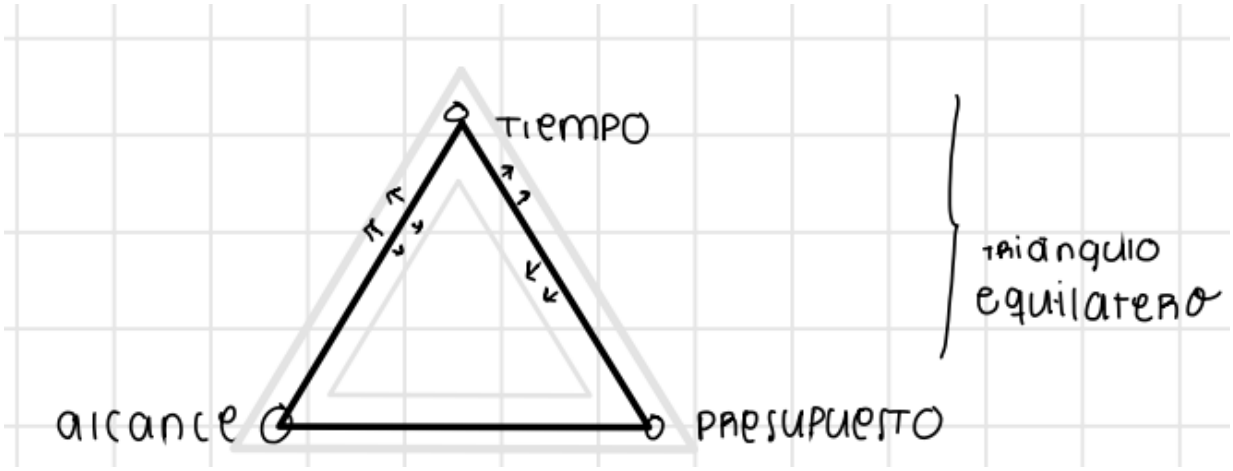
¿Qué es calidad de software?
- Software que hace lo que debe hacer.
- Software que tiene la estructura para ser mantenido y utilizado.

Fiabilidad: Se mide la posibilidad que el software falle con respecto a: production incidents, número de incidentes bajo presión y promedio de errores por módulo o usuarios.

Rendimiento: Se mide:

- Load testing: para entender el rendimiento con cargas de usuarios.
- Stress testing: para entender el límite de la plataforma.
- Soak testing: para entender hasta dónde puede aguantar la plataforma.
- Application performance monitoring: comportamiento bajo ciertos perfiles.

Capacidad de Entregar Software:



Análisis, Diseño e Implementación de software:

Estimación de tardanza en completar un requerimiento:
¿Que se estima?

Table Entry	Possible Entries
What's estimated	Size, Effort, Schedule, Features
Size of project	S M L (Small, Medium, Large)
Development stage	Early, Middle, Late
Iterative or sequential	Iterative, Sequential, or Both
Accuracy possible	Low, Medium, High

¿Qué se puede contar?

Quantity to Count	Historical Data Needed to Convert the Count to an Estimate
Marketing requirements	• Average effort hours per requirement for development • Average effort hours per requirement for independent testing • Average effort hours per requirement for documentation • Average effort hours per requirement to create engineering requirements from marketing requirements
Features	• Average effort hours per feature for development and/or testing
Configuration settings	• Average effort per configuration setting
Lines of code already written	• Average number of defects per line of code • Average lines of code that can be formally inspected per hour • Average new lines of code from one release to the next
Test cases already written	• Average amount of release-stage effort per test case

Change requests	• Average development/test/documentation effort per change request (depending on variability of the change requests, the data might be decomposed into average effort per small, medium, and large change request)
Web pages	• Average effort per Web page for user interface work • Average whole-project effort per Web page (less reliable, but can be an interesting data point)
Reports	• Average effort per report for report work
Dialog boxes	• Average effort per dialog for user interface work

Database tables	• Average effort per table for database work • Average whole-project effort per table (less reliable, but can be an interesting data point)
Classes	• Average effort hours per class for development • Average effort hours to formally inspect a class • Average effort hours per class for testing
Defects found	• Average effort hours per defect to fix • Average effort hours per defect to regression test • Average number of defects that can be corrected in a particular amount of calendar time

¿Cómo podemos calibrar?

APPLICABILITY OF TECHNIQUES IN THIS CHAPTER

	Calibration with Industry-Average Data	Calibration with Organizational Data	Calibration with Project-Specific Data
What's estimated	Size, Effort, Schedule, Features	Size, Effort, Schedule, Features	Size, Effort, Schedule, Features
Size of project	S M L	S M L	S M L
Development stage	Early–Middle	Early–Middle	Middle–Late
Iterative or sequential	Both	Both	Both
Accuracy possible	Low–Medium	Medium–High	High

Your estimates can be calibrated using any of three kinds of data:

- *Industry data*, which refers to data from other organizations that develop the same basic kind of software as the software that's being estimated
- *Historical data*, which in this book refers to data from the organization that will conduct the project being estimated
- *Project data*, which refers to data generated earlier in the same project that's being estimated

Individual Expert Judgement

Judgement is provided based upon a specific set of criteria acquired in a specific knowledge area.

	Estimated Days to Complete			
Feature	**Best Case**	**Most Likely Case**	**Worst Case**	**Expected Case**
Feature 1	1.25	1.5	2.0	**1.54**
Feature 2	1.5	1.75	2.5	**1.83**
Feature 3	2.0	2.25	3.0	**2.33**
Feature 4	0.75	1	2.0	**1.13**
Feature 5	0.5	0.75	1.25	**0.79**

Otras:

- Decomposition and Recomposition: Descomposición de los mejores y peores escenarios por funcionalidad.
- Estimation by Analogy: Esta funcionalidad se parece a la funcionalidad que hicimos el otro dia.

Subsystem	Actual Size of AccSellerator 1.0	Estimated Size of Triad 1.0	Multiplication Factor
Database	10 tables	14 tables	1.4
User interface	14 Web pages	19 Web pages	1.4
Graphs and reports	10 graphs + 8 reports	14 graphs + 16 reports	1.7
Foundation classes	15 classes	15 classes	1.0
Business rules	???	???	1.5

- Proxy-based estimates: Por aproximación, creemos que nos va a tomar _. No he hecho un cambio de contraseña pero si he hecho un ingreso de contraseña.
- Expert judgment in groups: Les platicas a los expertos, ¿Cuanto consideran que se van a tardar?

Planning:

Metodologías para estimar el trabajo que se debe realizar:

- Planning poker: 1,3,5,8,13
- T-shirt estimation
- Personas con mayor experiencia
- Promedios

CA:

Como profesora entro a la página principal y vo al botón de “Listas de Espera”:

- Al dar click, voy a una vista con todas mis clases.
- Al dar click en cada clase veo la Lista de personas inscritas y otro tab con “Lista de Espera”.
- Puedo seleccionar a las personas a Aceptar.
- Doy click en “aceptar”
- Se acepta la lista.

¿Cómo priorizar el backlog?

La actividad para determinar los requerimientos que se realizarán antes y las que se harán después.

Consideramos:

- Valor de negocio
- Return of investment
- Grupos de funcionalidades
- Política y equipo

Estrategias y técnicas para la prueba de software:

El proceso de analizar software para detectar las diferencias entre lo que se pidió y cómo funciona.

Tipos de Pruebas:

- Manual
- Automática

Métodos:

- Estático
- Dinámico

Formas:

- White box testing: revisar módulos, la estructura interna de la aplicación.
- Black box testing: no se nada del sistema solo checo que esté inputs y outputs. Se basa en evaluar la funcionalidad.
- Grey box testing: tipo sistema de alexa, conocer el convexo.

Niveles:

- Unit testing: revisar módulos.
- Integration testing: conexiones entre módulos. (microservicios)
- System testing: Escenarios completos.
- Acceptance testing: para lograr que el cliente acepte el servicio.

Tipos de Black box testing:

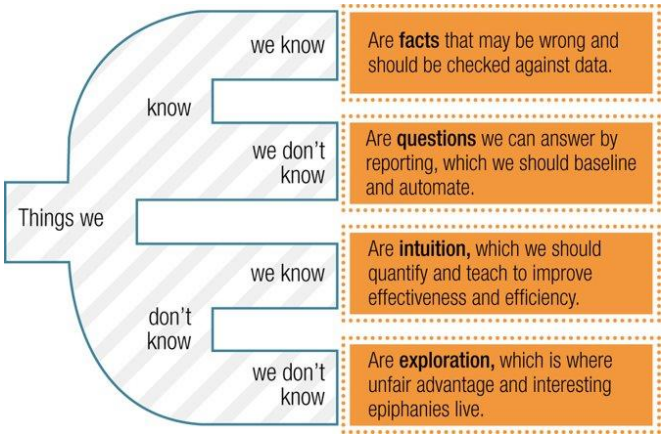
- Functionality testing:
- Non-functionality testing: ambiente en el que esta el sistema.

Artefactos:

- Tes plan
- Traceability matrix: lo que vamos a probar
- Test case
- Test script
- Test suite
- Test data or test fixture
- Test harness

Modelos de madurez en el proceso de software:

Métricas:



Características de buenas métricas:

- Comparables
- Entendible
- Proporciones
- Provoca cambios

Ejemplo: 10% de las inscripciones por semestre se hacen en 50% menos de tiempo con respecto al semestre pasado.

Tipos de métricas:

- Cuantitativas vs Cualitativas:
    - Cualitativas: no tienen una estructura, son anecdóticas y difíciles de agregar.
    - Cualitativas: se refieren a números duros, pero tienen menos descubrimientos.
- Métricas por vanidad vs con significado
    - Son métricas que nos hacen sentir bien pero no tienen un impacto en nuestro producto.
    - Número de visitantes de una página.
    - Número de likes en comunicación.
    - Número de cuentas registradas.
- Métricas de reportes y para explorar
    - Los reportes del día a día que no se enfocan en explorar la realidad de la aplicación, causan más daño que bien a nuestro producto.
- Métricas guías vs métricas que arrastras
    - Las métricas guías te dan expectativas y conocimiento para predecir el futuro, las métricas del pasado te alejan del conocimiento nuevo.
- Correlación vs Causalidad
    - Correlación: Cuando dos variables cambian de la misma manera.
    - Causalidad: Cuando una variable cambia a causa de otra.
    - Una correlación no implica necesariamente una relación de causa a efecto entre dos factores.

¿Qué queremos?

- Empatía
- Hacer pegajoso el concepto
- Buscar viralidad
- Buscar ganancias
- Escalar

Proposición de Budget:

COTIZAR UN PRODUCTO

22-OCT-2021

	A	B	C	D	E	F
	Alcance	Días	Días reales	Recursos	Capacidad	Precio
	Pantalla de Login					
	Pantalla de Login					
	Pantalla de Recuperar contraseña					
	Precio Operativo					
	Precio Total					

equipo:

	Salario	Horas Laborales	Horas Reales	Precio por hora
enior Software Developer	47500	160	120	296,875
rogram Manager Intermedio	50000	160	120	312,5
evOps junior	45000	80	60	562,5
IX intermedio	33000	80	60	412,5
II intermedio	35000	160	120	218,75
osto por equipo	210500	640	480	328,90625

Hay días reales:

22-OCT-2021

$ 2,597.12

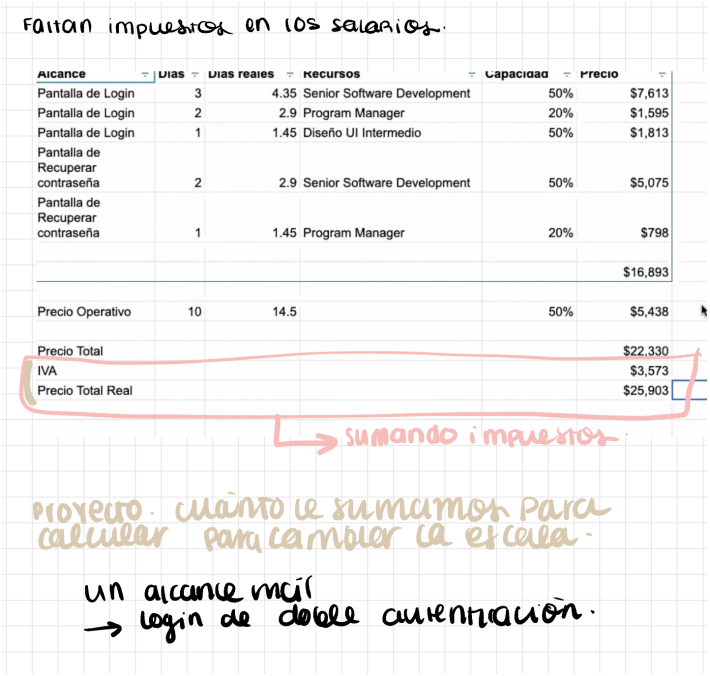
Presupuestos:

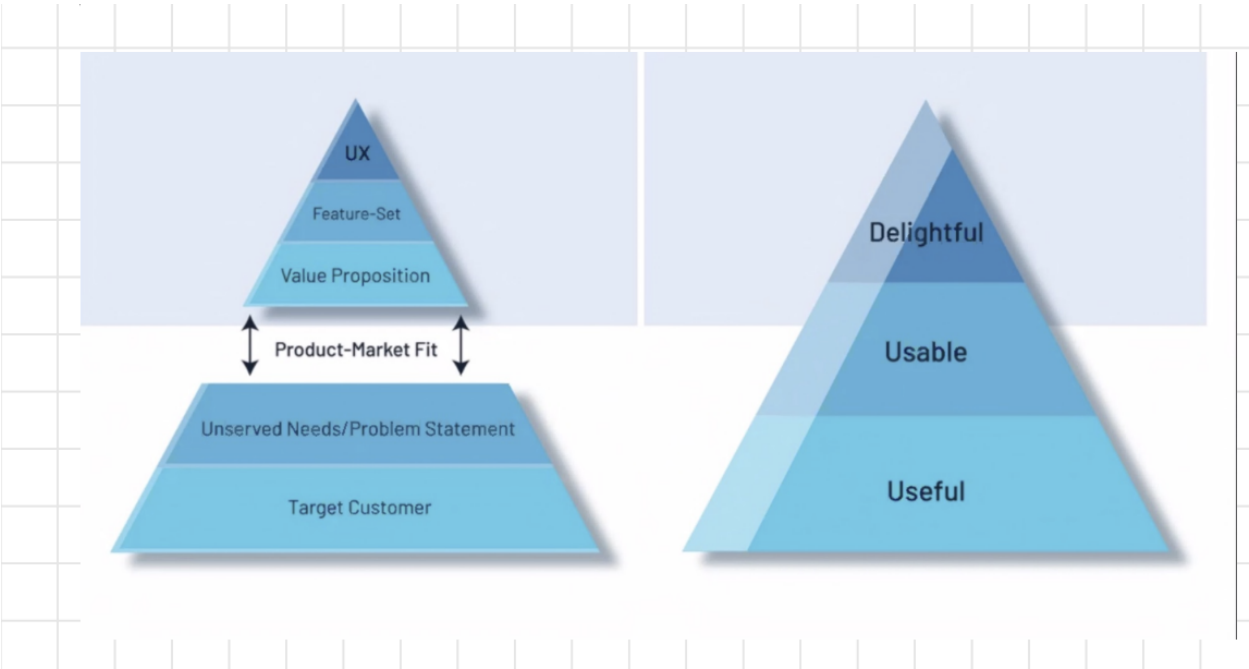
¿Por qué mi tecnología es la mejor"

Triángulo de la organización.

normalmente se estima de menos pero hay que pensar a onda

Alcance	Días	Días reales	Recursos	Capacidad	Precio
Pantalla de Login	3	4.35	Senior Software Development	50%	$7,613
Pantalla de Login	2	2.9	Program Manager	20%	$1,595
Pantalla de Login	1	1.45	Diseño UI Intermedio	50%	$1,813
Pantalla de Recuperar contraseña	2	2.9	Senior Software Development	50%	$5,075
Pantalla de Recuperar contraseña	1	1.45	Program Manager	20%	$798
					$16,893
Precio Operativo	10	14.5		50%	$5,438
Precio Total					$22,330





Mantenimiento de software:

Integración Continua:

Es la práctica de integrar continuamente el trabajo del equipo de ingeniería con los ambientes de producción.

Para que la integración continua sea posible se requieren de cuatro herramientas para comenzar. Repositorios de Código, ambientes de desarrollo variados, deployment automatizado y pruebas automatizadas.

Ambiente de Prueba:

Environment/Tier Name	Description
Local	Developer's desktop/workstation
Development/Trunk	Development server aka sandbox. This is where unit testing is performed by the developer.
Integration	CI build target, or for developer testing of side effects
Test/QA/Internal Acceptance	This is the stage where interface testing is performed. Quality assurance team make sure that the new code will not have any impact on the existing functionality and they test major functionalities of the system once after deploying the new code in their respective environment(i.e. QA environment)
Stage/Pre-production/External-Client Acceptance	Mirror of production environment
Production/Live	Serves end-users/clients

Deployment Automatizado: Docker daemon

Pruebas automatizadas: pruebas que se ejecutan de manera continua tan pronto una línea de código entra en el ambiente.

Componentes:

- Código y Proyectos
- Pruebas automatizadas
- Deployment
- Monitoreo de errores
- Integración continua
- Comunicación

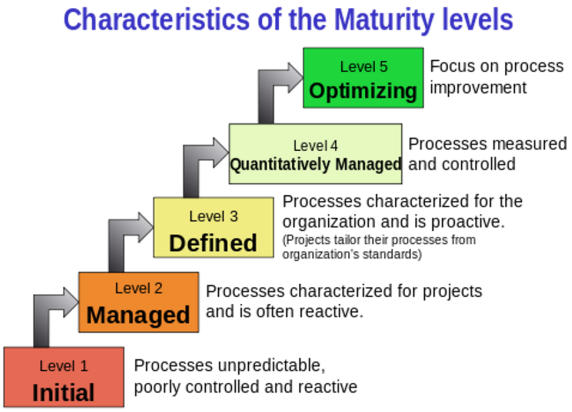
Escalamiento:

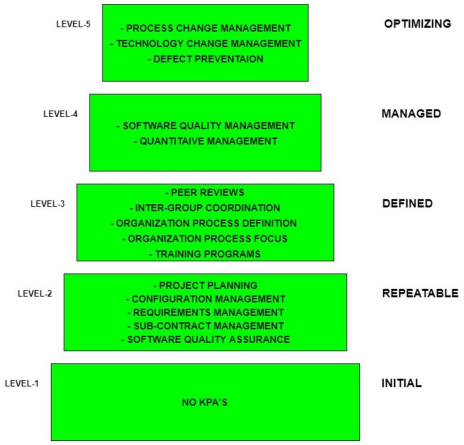
- El proceso de crecer el impacto de una aplicación en usuarios y/ transacciones.
- Vertical y horizontal.
- **Vertical:** Costo efectivo, comunicación menos compleja, mantenimiento menos complejo, no requiere cambios de software. Pero mayor downtime, un solo punto de falla y limitaciones de capacidad.
- **Horizontal:** sencillo de ejecutar desde el punto de vista del hardware, periodo corto de downtime, más resiliencia y resistencia a cambios, mejor performance. Pero mayor complejidad y mayor costo inicial.

Implantación en nube:

PSP/TSP/CMM:

Capability Maturity Model (CMM):





Personal Software Process (PSP):

PSP shows engineers how to:

- Manage the quality of their projects
- Make commitments
- Improve estimating and planning
- Reduce the defects

PSP framework activities:

- Planning
- High level design
- High level design review
- Development
- Postmortem: Using measures and metrics to improve.

Principios:
Every engineer is different; to be most effective, engineers must plan their work and they must base their plans on their own personal data.

- To consistently improve their performance, engineers must personally use well-defined and measured processes.

- To produce quality products, engineers must feel personally responsible for the quality of their products. SUperior products are not produced by mistake; engineers must strive to do quality work.
- It costs less to find and fix defects earlier in a process than later.
- It is more efficient to prevent defects than to find and fix them.
- The right way is always the fastest and cheapest way to do a job.

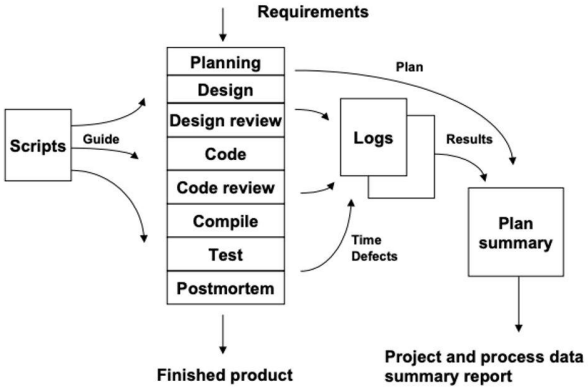


Figure 1: PSP Process Flow

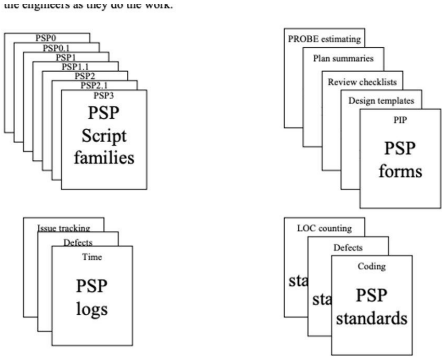


Figure 2: PSP Process Elements

Table 2: PSP1 Process Script

Phase Number	Purpose	To guide you in developing module-level programs
	Entry Criteria	• Problem description • PSP1 Project Plan Summary form • ***Size Estimating Template*** • ***Historical estimate and actual size data*** • Time and Defect Recording Logs • Defect Type Standard • Stop watch (optional)
1	Planning	• Produce or obtain a requirements statement. • ***Use the PROBE method to*** estimate the total new and changed LOC required. • ***Complete the Size Estimate Template.*** • Estimate the required development time. • Enter the plan data in the Project Plan Summary form. • Complete the Time Recording Log.
2	Development	• Design the program. • Implement the design. • Compile the program and fix and log all defects found. • Test the program and fix and log all defects found. • Complete the Time Recording Log.
3	Postmortem	Complete the Project Plan Summary form with actual time, defect, and size data.
	Exit Criteria	• A thoroughly tested program • Completed Project Plan Summary form with estimated and actual data • ***Completed Size Estimating Template*** • ***Completed Test Report Template*** • Completed PIP forms • Completed Defect and Time Recording Logs

Team Software Process (TSP):

PSP is necessary as a prerequisite.

PSP and TSP, help the high performance engineer to:

- Ensure quality software product
- Create secure software product
- Improve process management in an organization

TSP framework activities:

- Launch high level design
- Implementation
- Integration
- Testing
- Postmortem

¿Qué hace un buen equipo?

- A team consists of at least two people.
- The members are working toward a common goal.
- Each person has a specific assigned role.
- Completion of the mission requires some form of dependency among the group members.

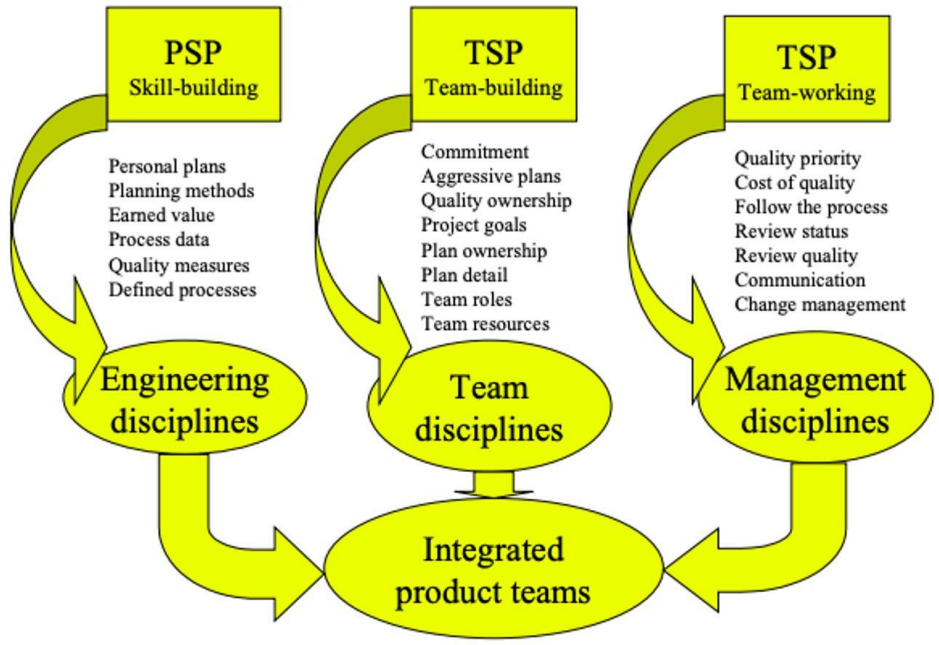


Figure 2: TSP Team-Building

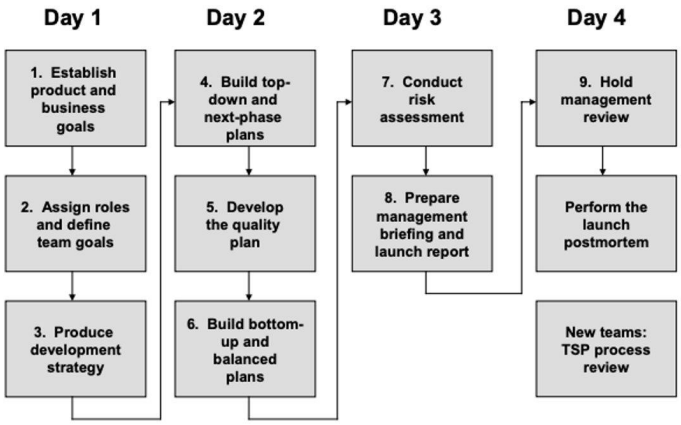


Figure 4: The TSP Launch Process