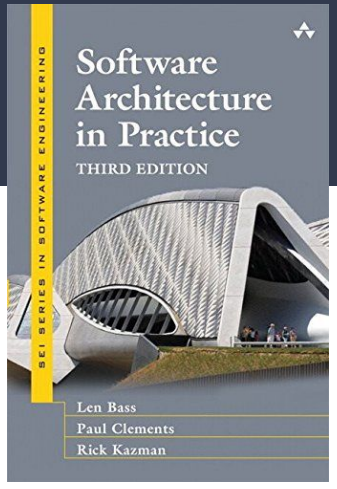
Tácticas - Performance, Scalability, Availability

Repaso, QAs y tácticas

1er cuatrimestre 2023

Recordando conceptos

Conceptos



Arquitectura de software

(de un sistema)

*Conjunto de **estructuras necesarias** para **razonar** sobre el sistema, que constan de **elementos** de software, **relaciones** entre ellos, y **propiedades** de ambos*

Atributos de Calidad

Requerimientos

- Funcionales
  - Lo que el sistema debe hacer
- Sobre Atributos de Calidad
  - Cualidades de los req. funcionales
- Constraints
  - Decisiones ya tomadas que no se pueden cambiar

Atributo de calidad

Propiedad mensurable ó testeable del sistema que es utilizada para indicar cuán bien el sistema satisface las necesidades de sus stakeholders

Táctica Arquitectural

Decisión de diseño que influencia la consecución de un atributo de calidad

La red conceptual ejemplificada

Objetivos del negocio

"Permitir realizar el 100% de los trámites administrativos de forma online"

Motivan

Requerimientos del sistema

RF1. Inscripción online a cursos
RF2. Admin online de cursos y materias
...
RQA1. Responder en menos de 2s
RQA2. Permitir 500 inscripciones por minuto
RQA3. Disponible días hábiles 8 a 20hs
RQA4. Facilitar la inscripción a través de nuevas apps cliente en el futuro

Se aplican sobre

Propiedades de Arquitectura

- Funcionales
  - UI y API de inscripción
  - UI de admin
- **Atributos de Calidad**
  - **Performance**
  - **Disponibilidad**
  - **Extensibilidad**

Promueven o inhiben

Tácticas

- Permitir concurrencia
- Múltiples web servers
- Redundancia activa de web servers
- Encapsular funcionalidad de inscripción en una API
- Modularizar con Layers

Determinan relaciones, responsabilidades y roles de los

Elementos

Estáticos: Módulos
- **UI Layer**
- **Business Logic Layer**
- **Data Access Layer**
- **Web module**
- **CLI module**

Dinámicos:
- Componentes: web **browsers, Load Balancer, Web Servers, FIUBA DB**
- Conectores: **protocolos HTTPS, TCP**
- Datos: **interfaces en HTML y JSON**

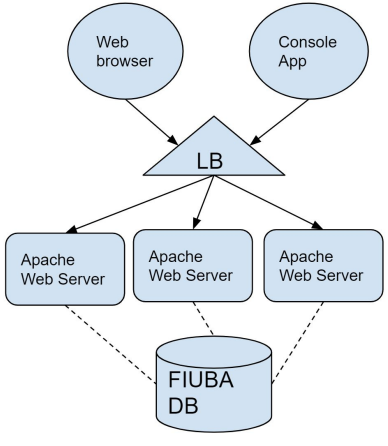
Arquitectura de software de un sistema

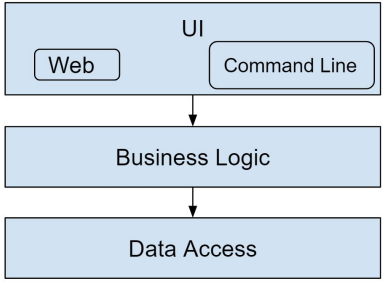
Estructuras

Estáticas: módulos
Dinámicas: C&C, Allocation

Se representan a través de

Vistas





Objetivos de Negocio ≠ Requerimientos

Ejemplos de Objetivos de Negocio

1. *“Ganar más porción del mercado”*
2. *“Mantener una reputación alta”*
3. *“Agregar nuevas funciones fácilmente”*
4. *“Proveer un framework amigable para programadores”*
5. *“Integrarse con otros sistemas fácilmente”*
6. *“Superar a la competencia en términos de velocidad de procesamiento”*

Atributos de Calidad sobre los cuales podría haber requerimientos

1. Modificabilidad, Usabilidad
2. Disponibilidad, Usabilidad, Performance
3. Modificabilidad, Disponibilidad
4. Modificabilidad
5. Interoperabilidad, Portabilidad, Modificabilidad
6. Performance, Escalabilidad

Atributos de Calidad

Propiedades del sistema en **runtime**

Performance

Scalability

Elasticity

Reliability

Availability

Visibility

Interoperability

Security

Usability

Propiedades del **desarrollo** del sistema

Modifiability

Portability

Simplicity

Testability

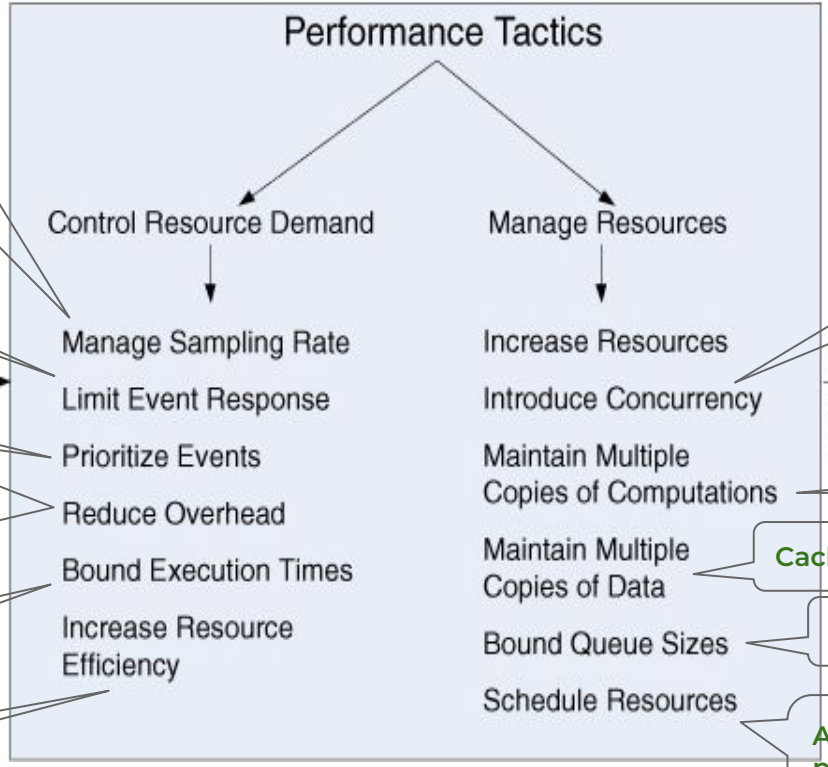
Performance

Scalability

Availability

Performance

Responder en tiempo a los eventos que se deben responder



Si se capturan datos del entorno

N tx/s. Encolar excedentes

Baja prioridad podrían ignorarse

Eliminar elementos intermedios. Co-location. Cleanup recursos ineficientes por su gran tamaño.

Limitar iteraciones

Mejorar algoritmos

Si el problema es paralelizable

Ej.: Más servers

Cache. Replicación para reducir contención

Limitar cantidad eventos encolados.

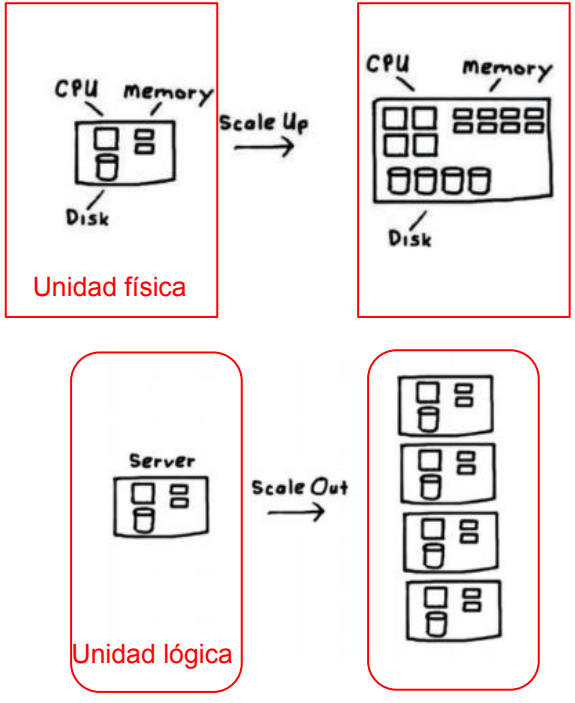
Acceso a recursos a través de políticas como FIFO, prioridad fija, prioridad dinámica (ej. round robin)

Fuente: "Software Architecture in Practice", 3rd Edition. Len Bass, Paul Clements y Rick Kazman. 2013.

Scalability

¿Esfuerzo?
¿Costo?
¿Interrupciones?
¿Es medible?

*Incorporar recursos adicionales de manera **efectiva***



Scalability Tactics

- Stateless
  - store state in the client
  - distributed state cache separated from web/app servers
- Scale out not up
- Avoid hard-coded resource assumptions or limits
- Design to be monitored
  - identify dead services
  - log files
  - evaluate end user response times
  - self-diagnosis
- Async Design
  - Desacoplamiento de componentes
  - Request puede ser temporalmente almacenado hasta que algún componente pueda procesarlo
- Sharding
- Design for multiple live sites
  - favorece Availability también

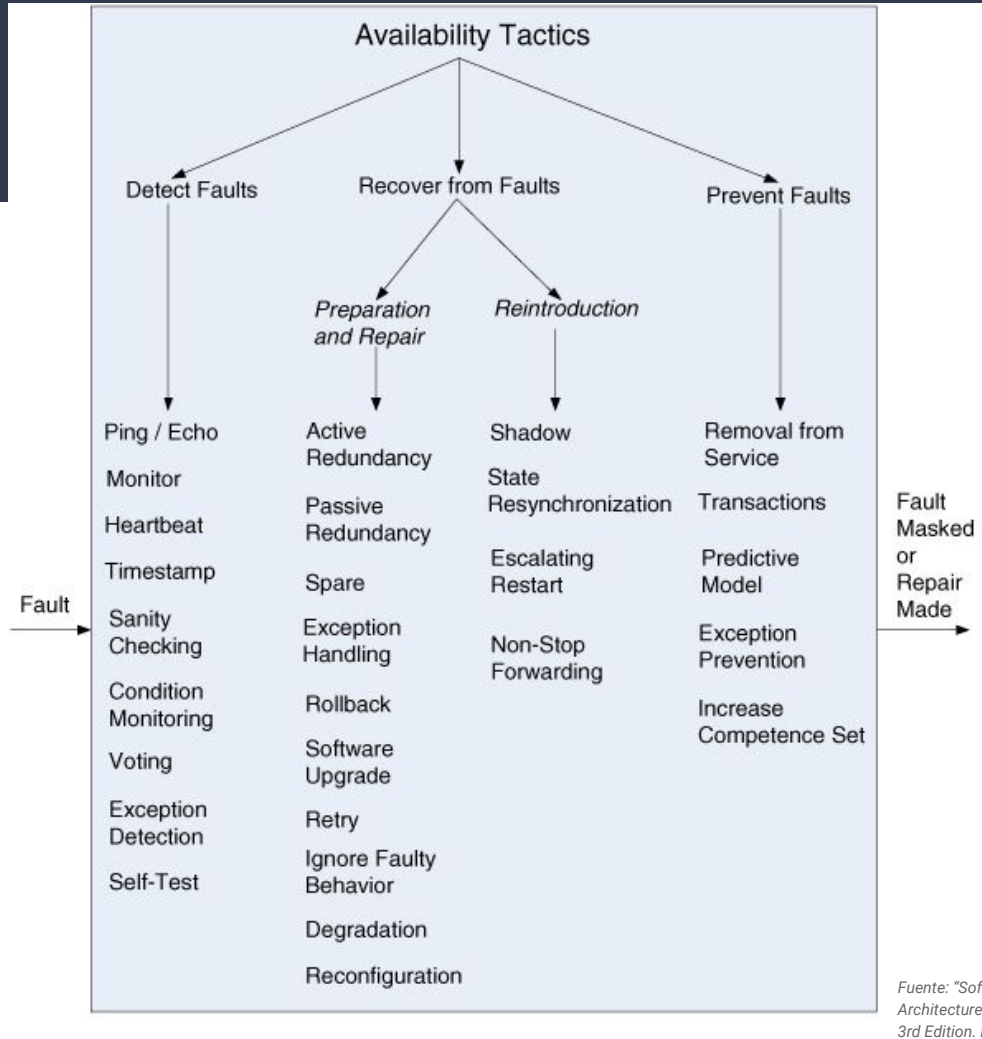
Availability

Reparar ó encubrir fallas de forma que el período fuera de servicio no exceda cierto valor

"Que el servicio esté levantado y listo para llevar adelante sus tareas cuando se lo requiere"

Reliability: *"Grado en que la arquitectura es susceptible a fallas a nivel de sistema cuando se producen fallas parciales en componentes, conectores, o datos."*

Availability = Reliability + recovery

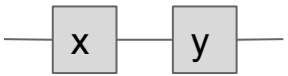


Fuente: "Software Architecture in Practice", 3rd Edition. Len Bass, Paul Clements y Rick Kazman. 2013.

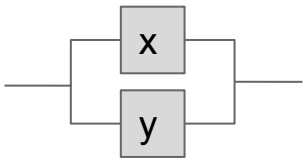
Availability

A = P(sistema provea servicios dentro de los límites requeridos)

A = MTBF/(MTBF+MTTR)



A = A(x)*A(y)



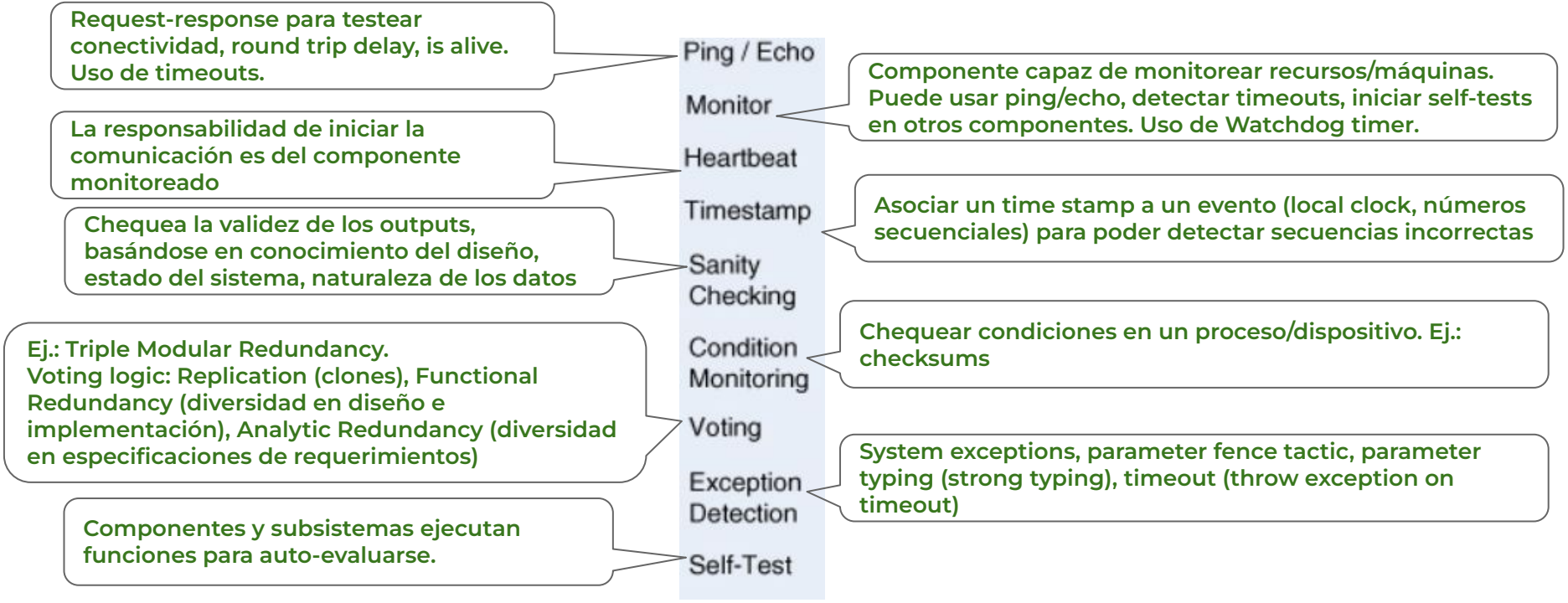
A = 1-(1-A(x))*(1-A(y))

Availability	Downtime/90 Days	Downtime/Year
99.0%	21 hours, 36 minutes	3 days, 15.6 hours
99.9%	2 hours, 10 minutes	8 hours, 0 minutes, 46 seconds
99.99%	12 minutes, 58 seconds	52 minutes, 34 seconds
99.999%	1 minute, 18 seconds	5 minutes, 15 seconds
99.9999%	8 seconds	32 seconds

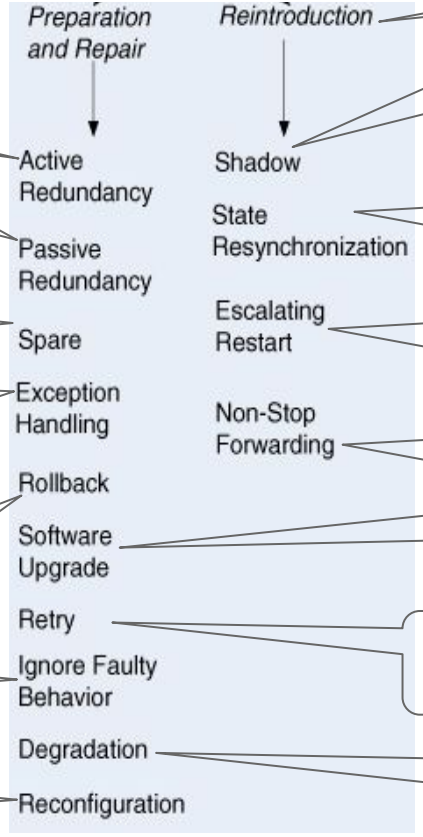
La pizzería “De la Esquina” decide publicar su página en la Web, y para eso desarrolla un site que, por el momento, está instalado en una única máquina Windows con IIS (Internet Information Services - software web server), y una pequeña base de datos MongoDB.
El desarrollador que mantiene el sitio es notificado cada vez que el sitio está caído. Él estima que, en promedio, lo llaman cada 45 días por este tema, y que desde el momento que el sitio experimenta la falla, en promedio transcurren:
12 horas hasta que alguien lo detecta y avisa a la pizzería, 24 horas hasta que lo contactan a él, 4 horas hasta que él puede dedicarse al problema, y 2 horas en arreglarlo y volver a poner el sitio en producción.

a) ¿Cuál es el “downtime” promedio por mes? ¿y por día?
b) Calcular el porcentaje de disponibilidad del sitio.
c) ¿Qué tácticas podemos aplicar para aumentar la disponibilidad?
d) En función de la respuesta del punto anterior, realizar una estimación de cuánto podría ser la nueva disponibilidad del sistema una vez aplicadas las tácticas (hacer las hipótesis que sean necesarias).

Availability - Faults detection



Availability - Recovery



Un componente que falló es vuelto a ser agregado luego de ser corregido

Todos procesan el input, en paralelo.

Se monitorea el componente corregido en estado "escondido" antes de re-introducirlo

Activos procesan input, y envían actualizaciones a los redundantes

Usado con Active/Passive redundancy

Redundantes fuera de servicio hasta que ocurra un fail-over

Reiniciar sistema en varios niveles

Capturar excepciones e info de contexto, de forma tal de enmascarar el defecto (ej. corrigiendo la causa y reintentando)

Usado en routers, dividiendo la funcionalidad en dos partes: control plane y data plane

Revertir estado del sistema a un estado estable anterior. Puede ser usado en conjunto con active/passive redundancy.

Actualizar código sin afectar el servicio.
- Function/class patches
- ISSU (In Service Software Upgrades)

Ignorar mensajes de ciertas fuentes que consideramos espúreas. Ej. DoS attack.

Para defectos temporales. Debe haber un límite, ante el cual se asume defecto permanente

Reasignar servicios a recursos que estén funcionando

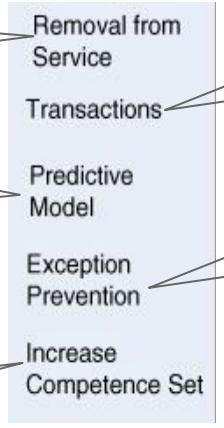
En presencia de defectos, mantener funciones críticas y dar de baja otras

Availability - Prevention

Poner un componente fuera de servicio temporalmente. Ej. reiniciar componente para limpiar defectos acumulados (memory leaks, fragmentación, etc.)

Combinado con Monitor, para monitorear la salud del sistema y tomar acciones en caso de predecir una posible futura falla.
Ejs.: session establish rate en HTTP server, threshold crossing, process state statistics.

Competence set: conjunto de estados en que el sistema puede operar de forma “competente”.
Diseñar para manejar más casos (defectos) como parte de la operación normal.
Ej.: si el acceso a un recurso está bloqueado, esperar un poco, o devolver una respuesta que indique que el procesamiento se terminará más tarde.



Utilizar transacciones ACID para async messages intercambiados entre componentes distribuidos. Ej. Implementación de 2-phase commit

Uso de exception classes, abstract data types y wrappers (ej. smart pointers) que manejen memoria y concurrencia.

¿Consultas?

Feedback:

http://bit.ly/arq-soft-feedback-clases

Guillermo Rugilo

guille.rugilo@gmail.com