UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)
FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)

Resiliència a nivell d'aplicació

Grau en Enginyeria Informàtica – Enginyeria del Software

Autor: Mihai Lucut

Director: Dimas Cabré Chacon, Everis

Ponent: Xavier Burgués, ESSI

Gener 2017

Índex

1 Gestió del projecte

1.1 Context

Poc després d'haver construït el que es considera la primera computadora[1] han començat els intents per fer que aquesta adopti el mètode de *pensament* humà o racional. Encara que com a àrea de la ciència només es va reconèixer una dècada després. Ens referim a la Intel·ligència Artificial[2], entre els principals problemes que tracta es troben: el raonament, el coneixement, la planificació, el processament de textos naturals i la percepció. Inclús l'arquitectura del pc ha anat evolucionant perquè s'assembli al que hem arribat a conèixer del nostre propi cervell.

La resiliència és, per tant, un altre exemple de qualitat pròpia de l'ésser humà que volem que caracteritzin els sistemes. Hollnagel[2] la defineix com l'habilitat intrínseca d'un sistema d'adaptar el seu funcionament abans, durant o després de canvis i disturbis de tal manera que mantingui disponibles les funcions requerides en condicions esperades o no.

En primer lloc es va començar a estudiar la resiliència en els sistemes informàtics en l'àmbit del hardware. Des de la redundància a nivell de bits o a nivell de

[1] L'Eniac, presentat al 1946 en Pennsylvania
[2] Hollnagel, E., Woods, D. D., & Leveson, N. (2006).Resilient Engineering - Concepts and Precepts(pp.11–13). Shgate

fitxers pel que fa l'emmagatzematge de dades, fins als mecanismes d'encaminament d'internet. El hardware pateix errors de tota mena per això es fan còpies de seguretat, utilitzem tecnologies de virtualització de l'emmagatzematge com els RAID, etc.

El terme resiliència en l'àmbit dels sistemes informàtics podria donar la sensació que implícitament es refereixi a aspectes hardware. Encara que més tard, la resiliència en l'àmbit del software ha arribat, ja que el software també pateix errors. No només degut al fet que s'està executant sobre el hardware. El software conté errors, com més complex un sistema més errors. (a més segons tal i cual, en un sistema complex no és que hi sorgeixin molts errors, es que sempre hi ha alguna part que esta patint algun error) Per tant, entenem per aplicació resilient aquella que segueix donant servei a un determinat nivell de qualitat després d'haver patit errors. Des del cas desitjat; que l'usuari no se n'adoni, passant per la degradació gradual, recuperar-se o inclús prevenir aquests errors.

L'objectiu d'aquest treball és estudiar i proposar principis que es podrien seguir, des del disseny, per aconseguir aplicacions resilient. Per tant, des del principi ja estem suggerint que el concepte és transversal tant a nivell de fase del desenvolupament com a nivell de rol. Un subconjunt dels principis proposats s'implementaran i s'analitzarà el seu funcionament i eficàcia mitjançant una aplicació mòbil.

El projecte ha estat impulsat pel departament d'Innovació d'Everis que és el principal *stakeholder*, promotor i beneficiari directe del projecte. El valor que el projecte aporta no resideix en ell mateix, sinó en la seva aplicació en futurs projectes. Els beneficis del software resilient, pensem que no són exclusivament per l'entorn de producció, tal com sosté Michel T. Nygard i Uwe Friedrichsen.

1.2 Estat de l'art

El pioner en utilitzar aquest concepte és en Michael T. Nygard. El seu llibre *Release It!*[3] encara que hagi sortit fa una dècada es considera com la bíblia de les aplicacions resilient. Estudia diversos principis que es poden implementar mitjançant paterns a fi d'aconseguir software resilient, llest per l'entorn de producció. En aquest sentit, a diferència d'en Michael, nosaltres pensem que el software en general avui en dia està massa pensat per l'entorn de producció. Considerem que l'esforç d'aconseguir resiliència es recompensa en tots els entorns i no només l'usuari final.

Al llarg del llibre es presenten patrons de disseny i s'analitzen, mitjançant casos reals de l'experiència de l'autor. En posteriors treballs i conferències sobre el tema es

[3] Release it! Design and Deploy Production-Ready Software, 2007, The Pragmatic Bookshelf

tornen a explicar els principis proposats per en Michael i també es recomana i referencia el seu llibre.

Un exemple és la presentació d'Uwe Friedrichsen que s'anomena *Patterns of resilience* que apareix al 2015. Ell fa un recull d'alguns patrons explicant-los però tenint el mateix enfocament: la resiliència està pensada per i només per la producció. Els patrons estudiats per ell, amb exemples més a prop de la tecnologia actual, estan sota el paraigües dels grans patrons del disseny de l'enginyeria software com ara: baix acoblament, aïllament, etc.

Alguns autors no parlen en aquests termes per aconseguir resiliència. Com és el cas de Nicolo Perino què, proposa crear un *framework* que doti sistemes de caràcter general amb capacitats d'auto recuperabilitat. L'ambició d'aquest *framework* és esquivar errors funcionals en temps d'execució. El seu enfocament es basa en la redundància intrínseca de les llibreries, és a dir, trobar mètodes independents que proporcionin la mateixa resposta.

No és l'únic framework dissenyat per a proporcionar resiliència i cada cop en surten més. Alguns ja estan consolidats i comercialitzats com per exemple akka, o hystrix. Altres, com els projectes Simian Army proven el nivell de resiliència i ajuden a

millorar-la. Partint de la idea de deixar un mico armat en un *datawarehouse*, s'han implementat diversos projectes que ataquin o inspeccionin instancies o clústers de forma aleatòria en els sistemes de Netflix. Tenint en compte el principi que menciona Michael Nygard; els cucs de llarga durada no es poden detectar en la fase dels tests. Netflix amb la Simian Army prova les seves aplicacions en l'entorn de producció. A la Taula 2 conté els projectes i la seva descripció.

Mico	Descripció
Chaos Monkey	Apaga instancies de manera aleatoria.
Latency Monkey	Introdueix demores artificials en la comunicació entre client i servidor.
Conformity Monkey	Apaga instancies que no segueixen les bones practiques.
Doctor Monkey	Notifica els serveis si troben instancies malaltes, per exemple massa CPU, i les apaga.
Janitor Monkey	Ordena i neteja l'entorn. Busca i esborra recursos no utilitzats.
Security Monkey	Busca violacions de seguretat o vulnerabilitats. Verifica la vigència dels certificats SSL i DRM.
10-18 Monkey	Detecta errors de configuració o d'execució de les instancies que serveixen clients en diferents zones geogràfiques.
Chaos Gorilla	Semblant al Chaos Monkey però simula un desbortament d'una zona de cobertura sencera (Amazon).

Taula 2. Projectes que composen el Simian Army de Netflix.

1.3 Formulació del problema

Conscienciar-nos que els errors poden passar desapercebuts en les fases de *debug* o de testeig va donar fruit a termes com *fault-tolerance, resilience, self-healing o anti-fragility etc.* Tots aquests conceptes parteixen d'una hipòtesi contraria a la clàssica en quant a maximitzar la disponibilitat d'un software. Mentre la manera clàssica era minimitzar el número d'errors, el nou enfocament dóna per suposat que els errors apareixeran i tracta de minimitzar el seu impacte. El canvi és en l'òptica de la coneguda fórmula de la disponibilitat:

$$\text{Availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

Formula 3. La disponibilitat d'un software.

Mean Time To Failure és el temps mitjà entre errors, com més gran sigui vol dir que hi ha menys errors, el que significa més disponibilitat. Però la segona variable és el *Mean Time To Repair* que és el temps mitjà que dura la reparació.

Segons el nostre punt de vista la disponibilitat d'una aplicació no va lligada a l'entorn de producció. Aquesta qualitat és desitjable i es pot aconseguir mitjançant la resiliència. En termes d'éssers humans que interaccionen amb l'aplicació podríem dir que la resiliència és pot aconseguir per mitja de l'empatia. En el cas de l'entorn de desenvolupament aquest és el desenvolupador, i en el cas de l'entorn de producció n'és l'usuari final.

Tenir empatia cap al desenvolupador, li pot facilitar molt la construcció del software. Per exemple, el framework Spring què en iniciar una aplicació s'han de carregar 400 *beans* comportà un temps d'espera totalment innecessari en un entorn de desenvolupament. Pensar que l'usuari d'aquest framework només necessitarà 5-10 beans per provar la funcionalitat que està provant és ser empàtic. Això podria resultar en un mode d'execució d'aquest framework amb una política de carrega *on-demand* dels beans. Encara que aquest no és un exemple de resiliència, utilitzar l'empatia podria arribar a trencar la dita: <<*En casa de herrero, cuchillo de palo* >>.

1.4 Abast

Des de l'àmbit teòric volem estudiar l'origen i l'evolució del concepte de resiliència en el software. Els principis proposats per diversos estudiosos del tema i els avantatges d'aplicar-los. Aquests principis poden aparèixer com a patrons de disseny que aportin funcionalitats valuoses pel desenvolupador, el tester, fins arribar a l'usuari final.

Dins l'abast del projecte també està donar exemples del comportament d'aquests principis a la pràctica. Concretament s'implementaran els patrons coneguts com: *Timeout i Circuit Breaker* per tractar el tema de les dependències d'altres serveis. Tindrem en compte els modes d'execució de l'aplicació, mode *online* i *offline*. Finalment s'implementarà algun mètode per dotar l'aplicació de capacitat per evolucionar de manera automàtica.

Queda fora de l'abast del projecte és la metodologia d'aplicar aquests principis. Aquest projecte suggereix principis per aconseguir, no té l'ambició de proporcionar ni ser una guia sistemàtica d'on ni com aplicar-los. Per manca de temps i altres recursos, no s'implementaran tots els principis estudiats i analitzats en la part teòrica d'aquest projecte.

1.5 Metodologia i rigor

En primer lloc s'ha de fer una tasca de documentació. Consultant diverses fonts d'autoritat reconeguda en el tema de la resiliència software. A part d'utilitzar fonts confiables sempre es contrastarà la informació extreta amb el director. Sempre mantindrem una actitud critica davant dels principis de resiliència trobats sigui quina sigui la seva procedència. Les anàlisis dels principis tant com els principis proposats es validaran amb el director en reunions de manera incremental.

En quant a la segona part, la implementació es farà utilitzant la metodologia *agile*, fent *sprints* setmanals i reunions amb el director. En cada reunió s'avaluarà la feina feta i la velocitat del sprint. També és presentaran les històries d'usuaris del *backlog* a implementar pel proper sprint.

2 Pla de projecte

2.1 Objectius

Amb la realització del projecte pretenem arribar al compliment de quatre objectius, tal com es pot veure a la Taula 2. Els primers dos estan en l'àmbit teòric i els dos següents en l'àmbit pràctic. Com ja hem dit, la resiliència software ja es porta tractant des d'una dècada. Hi ha gama amplia de principis ja proposats, sobretot en últims anys[4]. Pretenem resumir i analitzar quan creguem necessari els principis que ja s'han proposat en aquest tema. També ens proposem aportar alguns principis propis i finalment volem implementar i estudiar els beneficis d'aplicar aquests principis al software.

Id	Objectiu
Obj1	Estudiar l'estat de la resiliència en l'àmbit del software.
Obj2	Resumir i analitzar principis de resiliència.
Obj3	Proposar nous principis de resiliència des d'un enfocament diferent.
Obj4	Implementar i analitzar els avantatges d'aplicar els principis proposats.

Taula 2. Objectius teòrics i pràctics del projecte.

[4] La majoria de les fonts utilitzades daten del 2015 o 2016.

2.2 Tasques

El projecte té una duració total estimada de vuit mesos i mig. Iniciat el 15 d'Abril 2016 i amb data final 31 de Desembre 2016. Amb un marge de quinze i vint dies per possibles desviacions que pot patir. Amb una càrrega total aproximada en hores de 735, que es dividiran en dos. La primera és teòrica i consisteix a estudiar, analitzar i proposar principis de resiliència per a aplicacions. La segona part és pràctica i consisteix a implementar un subconjunt dels principis analitzats en la part teòrica.

En l'annex s'adjunten els diagrames de Gantt amb l'estimació temporal i les relacions de precedència. Ja que només hi ha un autor, les dues fases i les tasques que les componen seran dutes a terme de manera seqüencial. Per aquesta raó no s'inclou el diagrama de Pert. El camí crític conté totes les tasques.

2.3 Principis teòrics de resiliència

En primera fase s'havia de cercar informació. La iniciativa partia del projecte anomenat Chaos Monkey, el codi del qual Netflix havia alliberat el 2015. Aquest servei forma part d'un conjunt de serveis que Netflix fa servir per provar la resiliència dels seus sistemes. D'aquí ha sorgit la idea de baixar un esglaó més, dins del que és la resiliència software, es volia estudiar els principis que podrien regir una aplicació *resilient*.

Així és com va començar la recerca de la informació en aquest àmbit. Un els primers recursos trobats va ser: *Release It*, un llibre de 350 pàgines, que es va llegir per complet. A més d'això ens hem proposat cercar informació en diverses bases de dades especialitzades com ara Goolgle Scholar, ACM, IEEE Xplore.

Per acabar de posar en marxa el projecte també ha començat un període per cercar l'aplicació que faríem servir per provar els principis de resiliència estudiats i/o proposats. El resultat d'aquesta tasca ha tingut un gran impacte sobre totes les tasques que conté la part d'implementació.

Aquesta tasca i tenint en compte que tots els recursos a llegir són digitals. Preveiem que ocuparà entre un 50-65% de tot el temps dedicat a la part teòrica del projecte. És una tasca important i necessària sobre tot si es vol aportar valor en qualsevol tema.

2.4 Aplicació dels principis de resiliència

Hi ha diverses raons per implementar una aplicació i aplicar els principis teòrics. Per tant hem decidit utilitzar els artefactes produïts en una assignatura anterior. Consta d'una aplicació Android que disposa d'un servei al qual accedeix mitjançant una API rest. El servei s'encarrega principalment de la persistència. L'aplicació consisteix a proveir informació del nivell d'adaptabilitat dels llocs o locals públics per persones amb discapacitats motrius.

Per raons que no rellevants s'ha hagut d'incloure la implementació de part de l'aplicació en el que és el projecte. Tot i que implementar una aplicació Android no és l'objectiu d'aquest projecte és una tasca necessària per acabar provant la viabilitat i el funcionament d'aquests principis. La implementació, per tant, de l'aplicació i aquests principis serà la tasca més important en temps de la segona part del projecte. El programador està assabentat dels requeriments i inclús amb el disseny a seguir. Per tant, pel que fa a l'aplicació, la part més costosa en temps serà la implementació, ja que serà la segona aplicació Android executada pel programador.

Per tant, les tasques de la segona part són principalment les fases de desenvolupament d'un projecte de software; Requisits, Disseny, Implementació i

Verificació. D'aquestes tasques no comentarem res més aquí, sortiran directament en el diagrama de Gantt.

2.5 Desviacions

Comentem breument les tasques que presenten riscos de provocar desviacions del pla temporal que hem fet.

En la tasca de la recerca, el mateix concepte de resiliència representa i ha suposat una desviació. Aquest concepte s'entén i s'ha aplicat abans en el món de l'enginyeria. S'utilitza en arquitectura en plans de contingència en casos de desastres naturals. En l'àmbit informàtic, la resiliència hi apareix en temes de xarxes i a nivell hardware. Aquesta ambigüitat i l'escassa informació sobre el concepte en l'àmbit del software fa que la tasca de recollida d'informació requereixi un esforç més gran i d'un estudi més detallat de paraules i conceptes claus relacionats.

En la tasca de desenvolupament de l'aplicació, concretament en la fase d'implementació preveiem dues potencials causes de desviacions. En primer lloc el fet que el programador no té experiència en desenvolupar aplicacions Android. La segona causa és el risc de canvi de tecnologia.

Per les possibles desviacions s'han deixat dues setmanes de marge per cada part, tal com es pot apreciar en els diagrames de Gantt. A més d'això durant el període de l'1 d'Agost al 16 de Setembre s'ha decidit fer un curs d'Android de 30 hores, justament per la falta d'experiència del programador. Encara i no tenir prou amb les dues setmanes de marge es reduiria l'abast en deixar fora de la implementació alguns principis.

2.6 Identificació dels costos

2.6.1 Costos directes

El cost total del projecte vindrà donat per la suma dels costos dels recursos consumits per cada fase. Tenint en compte els riscos associats a cada fase i la contingència. En primer lloc considerem els costs dels recursos humans tal com es pot apreciar a la Taula 4.

Rol	Cost per hora [€]
Cap de projecte	30,00 €
Analista	30,00 €
Dissenyador	30,00 €
Tècnic programador	25,00 €
Tester	25,00 €
Documentador	30,00 €
Tècnic de sistemes	25,00 €

Taula 4. Preu per hora del rol.

Afegint el nombre d'hores de cada rol i els percentatges de participació de cada rol en cada fase obtenim el cost per rol dels recursos humans. El cost per cada rol en el projecte i el cost total dels recursos humans es troben a la Taula 5.

Tasca	Hores	Rols	Cost
Fer recerca	67,5	Documnentador 90%, Cap de projecte 10%	2.025,00 €
Llegir i estudiar	67,5	Documentador	2.025,00 €
Resumir i criticar	67,5	Documentador	2.025,00 €
Proposar	90	Cap de projecte	2.700,00 €
Requisits	90	Cap de projecte	2.700,00 €
Disseny	67,5	Dissenyador 90%, Analista 90%	2.025,00 €
Implementació	157,5	Tècnic programador	3.937,50 €
Verificació	67,5	Analista 5%, Tècnic programador 5%, Tester 90%	1.704,38 €
TOTAL	675		19.141,88 €

Tabla 5. Cost dels rols en cada fase.

Tal com s'ha previst s'ha deixat un marge de dues setmanes, que donen la diferència d'hores entre el projecte total de 735 i les hores totals de cada rol per fase de 675.

Quant al risc: el canvi de tecnologies en la fase d'implementació, poc probable. En aquest cas, ens costaria una setmana més. Per aquesta setmana hem de sumar 100,00 € que seria el 10% del cost del Tècnic Programador.

2.6.2 Costos indirectes

Com a costos indirectes tenim l'amortització dels recursos hardware i l'energia elèctrica. És a dir, un portàtil: Dell i5-5300U a 2,3GHz i 8GB RAM i un terminal Android: Oneplus X Snapdragon 801 i 3GB de RAM. El cost total dels dispositius hardware és de 1570,00 €. La duració total de 32 setmanes equival a 224 dies de feina. Això dóna un cost d'amortització (1570 * 0,25 *224*4,5)/(8*365) = 135,49 €.

A més a més hem de considerar els 8 mesos de corrent i connexió a internet, recursos també necessaris per dur a terme el projecte. Segons el model del portàtil tenim un consum d'energia de 0,92[6] kWh, donant un total de 676,2 kWh pel total del projecte. Per tant, el consum total d'energia suposa un cost de 87,91 €[7]. El cost associat a la connexió a internet suposa un total de 8*50 = 400 €.

Al total de 19.141,88 + 56,20 + 135,49 + 87,91 + 400 = 19,821,48 € li hem d'afegir la contingència, en aquest cas d'un 12%. Per tant, el cost total del projecte arriba a 23,785.78 €.

2.7 Viabilitat econòmica

Cal mencionar que el projecte està concebut per aportar un coneixement que aportarà valor a l'empresa de cara als projectes futurs als quals s'apliqui. En conseqüència el projecte en si no s'espera que sigui viable econòmicament parlant. El fet d'implementar les aplicacions amb principis de resiliència, suposa per una banda un *overhead* pels equips de desenvolupadors, però, per l'altre banda implica un manteniment més barat. El cost afegit per la formació dels equips més el cost de produir una aplicació *resilient* surt en benefici de l'empresa.

2.8 Control de gestió

Al final de setmana es recolliran les dades preses cada dia, de la dedicació de cada rol per dia i tasca. Segons aquestes dades es comprovarà no només la quantitat d'hores dedicades sinó també el rendiment. D'aquesta manera es podrà estimar si el temps restant amb la productivitat actual portarà a acabar la tasca en el temps previst.

Per una altra banda, també, s'estimarà el cost d'aquestes hores i recursos utilitzats, tal com es pot observar en la secció de la Taula 6. Les possibles desviacions dels costos indirectes s'avaluaran al final de cada mes.

Tasques \ Desviacions		Tarifa		Consum		Total	
		Mà d'obra	Recursos	Mà d'obra	Recursos	Mà d'obra	Recursos
Fer recerca	Estimat						
	Real						

Taula 6. Control de gestió del desviament per tasca.

2.9 Sostenibilitat

2.9.1 Econòmica

Existeix una avaluació dels costos, tant dels recursos humans com els materials. El projecte es podria realitzar en menys temps i amb menys recursos, ja que l'autor no té l'experiència en cada rol que ha hagut d'ocupar en les distintes fases del projecte. Amb una persona amb més experiència o amb un equip especialitzat s'aconseguiria abaixar el cost del projecte. En la tasca de la implementació, una part considerable consisteix a desenvolupar l'aplicació, que no és tan rellevant en el projecte, és a dir que es podria haver reutilitzat altres aplicacions. El projecte està fet a demanda d'una empresa i es preveu la reutilització del treball en futurs projectes.

2.9.2 Social

El projecte aporta un valor afegit en l'aspecte social dels desenvolupadors que seguiran els principis *resilient*. Aquest valor, s'aconsegueix a curt termini amb beneficis per l'equip. Millora l'ambient de l'equip a causa de l'empatia mútua que facilita el desenvolupament del software. Per altra banda, l'impacte social d'aplicar principis de resiliència a les aplicacions també cobreix una necessitat de l'usuari final. Aquest té una millor experiència d'usuari, ja que l'aplicació en gran part s'encarrega de resoldre els problemes que podrien sorgir.

2.9.3 Ambiental

El principal recurs material utilitzat és el portàtil. Aquest té un fort impacte ambiental. Encara que és poc probable que arribi a ser o reciclat al final de la seva vida útil, encara servirà per la realització d'altres projectes. El portàtil, doncs, comporta un impacte ambiental negatiu encara que durant la realització del projecte no es produeixi un considerable volum de CO_2. L'energia elèctrica és la principal font i és la causa de la producció de mitja tona[8] de CO_2 durant la realització del projecte. Per una altra banda es preveu un ús reduït del paper que tindrà una generació de CO_2 menyspreable.

3 Principis de resiliència

Amb l'avanç del camp de la Intel·ligència Artificial és tornen a qüestionar preguntes bàsiques sobre el software. Com per exemple on està la línia entre el que es computable i que no. Per suposat, s'encarrega de més problemes, que d'alguna manera són inèdits. Però no només això sinó també totes les qüestions i característiques inherents al software. La fiabilitat, el manteniment, la disponibilitat etc. Aquest avançament ha trobat finalment reconeixement i vigor a causa dels descobriments dels límits què implica un enfocament determinista.

El present treball doncs, tracta principalment d'afegir resiliència a aplicacions deterministes. Encara que, tal com entenem nosaltres, la resiliència és podria aplicar de forma més natural utilitzant Machine Learning. Per tant, farem algun comentari sobre algun principi de resiliència que es podria aplicar en aquest cas.

La principal font de resiliència fins ara n'és, sense dubte, l'experiència. Aquesta és inherent i varia en funció de cada persona implicada en el desenvolupament del software. Un dels beneficis que dona l'experiència és la resiliència. Com ja hem mencionat, entenem per resiliència la capacitat del software a respondre en circumstàncies adverses i/o sortir d'elles de manera autònoma.

De la resiliència en l'àmbit del software no fa tant 2007[5] que se'n parla, gairebé una dècada. En el seu llibre, *Release it*, Michael T. Nygard tracta el tema de la resiliència software des de la seva experiència.

Cal dir que el terme de resiliència ha anat evolucionant. Hi ha tres termes que estan molt relacionats, veure Figura 1. El primer és el concepte de *Fault Tolerance*, que consisteix en construir software robust. Es a dir, la fallida d'un o més components no comporta la caiguda de tot el sistema. El tercer és el concepte d'*Anti Fragility* que és la capacitat d'un sistema no només de tornar a l'estat normal sinó avançar cap a un estat millor.

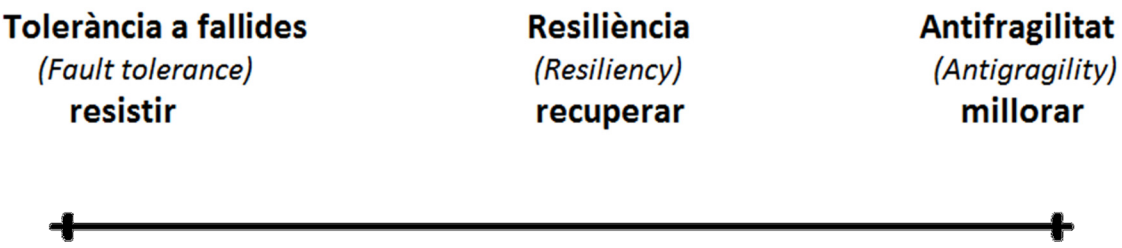


Figura 1. Mapa de conceptes

[5] Considerem el llibre Release it! com el primer en el que no només es parla sino que es tracta la resiliència software àmpliament i en detall.

En el llibre Release It, la resiliència esta més a prop de la tolerància a fallides. En general, els principis que proposa doten al software de resistència que ell considera com a resiliència.

Jonas Bonér, creador del framework akka[6], en la seva presentació[1], utilitza els termes de la Figura 1 per definir la resiliència. Per ell, la resiliència esta en algun punt entremig, és més que la Tolerància a fallides i menys que l'antifragilitat. Els principis de resiliència han de recuperar l'estat òptim. En quant a la resiliència purament software repeteix els patrons que apareixen en Release It. En temes de resiliència a nivell de sistema distribuït es centra en la resiliència que aporta akka, donant exemples d'ús.

Uwe Friedrichsen, també tracta el tema en algunes presentacions. En la primera[2], està d'acord amb Jonas Bonér afirmant que la resiliència va més enllà del fet de simplement resistir als errors, és tornar a l'estat òptim.

En canvi en la segona[3], hi ha una evolució en el concepte de resiliència. Aquí, proposa una arquitectura conceptual de la resiliència que comença a incloure el concepte d'antifragilitat.

[6] Akka és un conjunt d'eines (toolkit) i runtime per a la construcció d'aplicacions distribuïdes amb alta concurrència, resilient i orientades als missatges en JVM.

Nosaltres ampliem el concepte, anomenem principi de resiliència qualsevol principi que doti el software amb capacitat de resistència, recuperació o inclús millora. Es a dir, una aplicació resilient és aquella que segueix donant servei a un determinat nivell de qualitat després d'haver patit errors i és capaç de tornar a l'estat òptim o incús a un estat millor. En el cas ideal aquesta recuperació és transparent de cara a l'usuari. En els altres casos s'informa l'usuari que la funcionalitat no esta disponible temporalment i es torna a informar quan s'hagi efectuat la recuperació.

Considerem com activador del principi qualsevol error; passat, present o futur. Per això a la no hi ha cap punt per la resiliència,

3.1 Release it

Ara analitzarem els patrons que Michael T. Nygard proposa en el seu llibre, *Release it*[4], per aconseguir aplicacions resilient o més resilient. Alguns d'aquests patrons ja s'han anat incorporant als *bons costums* i/o a *frameworks* corresponents. Des del principi deixa clar que la principal motivació en construir software resilient és econòmica. El subtítol de la portada ho indica: "*Design and Deploy Production-Ready Doftware.* Segons sosté, una decisió de disseny és una decisió econòmica; i qualsevol *estalvi* que es vulgui fer en aquesta fase tindrà repercussions *cares* en producció. Per

tant, la programació ha de ser pragmàtica, orientada a l'entorn de producció, no a l'entorn de proves o QA.

Encara que és difícil trobar-se dos vegades amb el mateix problema, tard o d'hora surten els anti-patrons. Són aquelles situacions sistemàtiques que porten a errors, i per tant es poden aplicar solucions generals. El llibre s'estructura en quatre grans parts: Estabilitat, Capacitat, Reptes generals de disseny i Operacions. En els primers dos temes s'estudia en profunditat els problemes que provoquen els anti patrons i els patrons corresponents a les solucions.

En els darrers dos temes més que principis de resiliència són consells. Aspectes importants a tenir en compte a l'hora de dissenyar, com ara, la xarxa, la seguretat o la disponibilitat. Finalment, en el tema d'operacions tracta els aspectes de transparència i d'adaptació. Encara i estant enfocat només en l'entorn de producció el llibre aconsegueix donar una visió prou completa del patrons que es podrien aplicar per aconseguir software resilient.

3.1.1 Estabilitat

Com ja havíem dit, el primer tema que tracta és l'estabilitat. El software resilient ha de ser estable. Un error d'una certa funcionalitat no pot ser que ens faci

caure tot el sistema, deixant-nos sense poder fer res més abans de reiniciar l'aplicació o el servidor. L'autor identifica una gran varietat d'elements com a anti patrons en aquest tema. Aquests són: els punts d'integració, les reaccions en cadena, cascades d'errors, els usuaris, *threads* bloquejats, atacs d'auto denegació de servei, efectes d'escalat, capacitats no balancejades, respostes lents, SLA, respostes no determinades.

3.1.1.1 Anti patrons

Els punts d'integració es van multiplicant conforme el sistema d'informació d'una organització va creixent. Cada cop hi ha més fonts i consumidors d'informació que es necessiten integrar, necessiten interaccionar. Per exemple, CRM, ERP, MRP, BPO entre d'altres. Per tant cada socket, procés, pipe o crida remota pot i arribarà a penjar-se.

Les reaccions en cadena tenen que veure amb temes d'escalabilitat a nivell horitzontal. La Figura 1 mostra una granja amb vuit servidors darrere un balancejador de carrega. El problema apareix en cas de caiguda d'un servidor, els que queden s'han de repartir entre tots la seva feina. Depenent del tipus, l'error podria provocar la caiguda d'una altre servidor, fins arribar a caure tot el sistema.

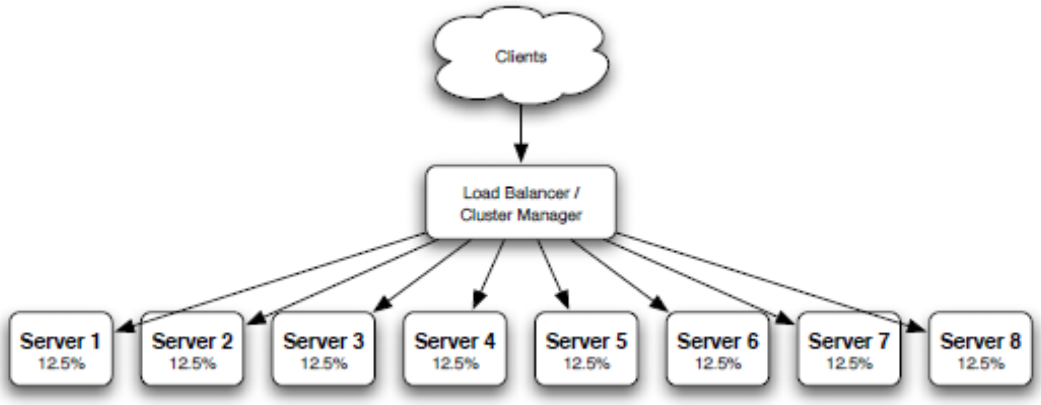


Figura 1. Exemple d'escalabilitat horitzontal.

Les cascades d'errors són semblants a les reaccions en cadena però a nivell de capes. Si els errors d'una capa provoquen erros en la capa que els crida parlem de cascades d'errors. Per exemple si el clúster de bases de dades es cau, i les aplicacions que servia no maneguen bé aquests errors aquestes començaran a fallar. Solen aparèixer quan s'esgota alguna pool amb recursos.

El comportament dels usuaris, tant de manera individual com general és prou demandant per no dir totalment imprevisible. A més el sistema escala en funció del hardware contractat i no en funció de la quantitat d'usuaris[7]. Per tant la pregunta és com reacciona el sistema quan la demanda supera la seva capacitat per respondre?

[7] Aquest és un exemple d'argument antiquat ja que des del segon trimestre del 2008 han començat a aparèixer serveis de host que proporcionaven un escalat en funció del nombre d'usuaris.

Els threads bloquejats apareixen a l'hora d'explotar el paral·lelisme de les CPU's. El multithreading és complex i normalment no és factible provar l'aplicació amb un nombre suficientment alt de requests. Per tant son problemes que difícilment surten abans d'entrar en producció.

Els atacs d'auto denegació de servei: *self-denial attack* apareix quan el sistema com un tot, inclús els humans "conspiren" en contra d'ell mateix. Per exemple una campanya de màrqueting que atreu molts més clients dels que el sistema esta preparat per rebre.

Les capacitats no balancejades tenen a veure amb el gestor d'escalat i les diferències entre recursos frontend versus backend.

Les respostes lents apareixen normalment quan el sistema ja esta en un nivell de demanda excessiu, per culpa del *garbage collector* o *memory leaks*.

El service-level agreement és el contracte que regula les condicions de servei. També conté les clàusules de penalitzacions econòmiques en cas que el servei no compleix les solucions. El problema és que un sistema no pot tenir un SLA millor que el de la pitjor de les seves dependències.

S'ha de dissenyar amb escepticisme. En molts casos una aplicació tracta la seva base de dades amb massa confiança. Qualsevol dependència pot en un moment donat retornar una resposta no esperada. Per exemple la base de dades podria respondre amb un resultat considerablement més gran que normalment. Si l'aplicació no limita la quantitat d'informació que esta disposada a processar poden passar coses no desitjades, el temps que triga és massa i l'usuari perd l'interès, desbordaments de memòria, etc.

3.1.1.2 Patrons

Per prevenir els escenaris problemàtics, en quant a l'estabilitat del sistema, enumerats més a dalt, Michael Nygard proposa vuit patrons. Com ja hem mencionats alguns ja estan implementats per les llibreries que és veuen actuant en dites circumstàncies. Per exemple, el primer patró és el timeout. Avui dia aquest principi ja està implementat en les llibreries, encara i així s'ha de ser conscient i configurar-ho pròpiament.

El següent patro s'anumena circuit breaker. Consisteix en monitoritzar el timeout i obrir el circuit si aquest salta molt sovint. Per tant, si el circuit està obert, ja sabem que no aconseguirem resposta, podem respondre que molt ràpidament. Un procés addicional es necessari en aquest cas per comprovar si el servei torna a estar

disponible. De manera automàtica, l'aplicació pot detectar això i tancar el circuit tornant a l'estat normal.

Els *bulkheads* o mampares (veure Figura 3), separen l'espai una embarcació en compartiments. En cas de produir-se forats, el compartiment afectat es pot tancar i contenir la propagació de l'aigua a la resta del vaixell. Seguint aquest exemple l'aplicació hauria d'estar dividida en particions que no deixin propagar els a traves de les mampares.

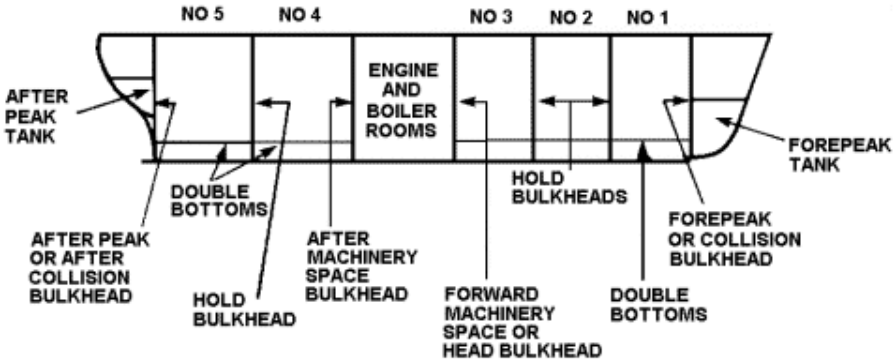


Figura 2. Mampares d'una embarcació.

Com a exemple tenim a Baz com a dependència de Foo i de Bar. Per exemple un manteniment Baz seria impossible de realitzar degut a la impossibilitat de respectar els SLA de Foo i SLA de Bar a l'hora. En aquest cas, Baz hauria d'estar compartimentat protegint els clients. Evidentment s'ha d'estudiar bé la la mida dels compartiments, des les thread pools fins als servidors en un claster.

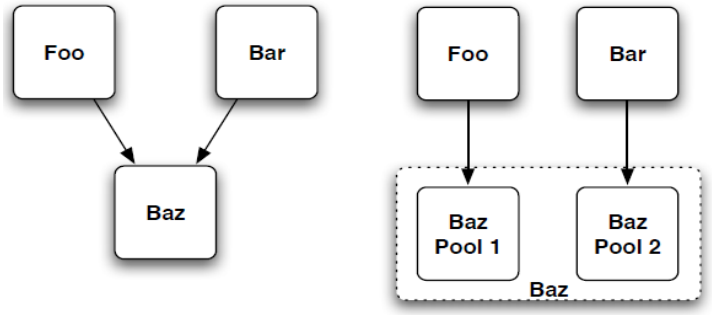


Figura 3. Aplicació del principi de mampares

El Steady-state és l'estat normal de l'aplicació. Aquest s'hauria de mantenir per si mateix sense necessitat d'intervenció humana diària. Pels problemes d'espai dels logs o neteja de la base de dades s'haurien de fer scripts que s'executin automàticament. Un altre aspecte a considerar per garantir un estat òptim de l'aplicació consisteix en controlar la memòria que la cache pot ocupar. Per últim els logs. Aquests, si s'han de conservar per llei és recomana no mantenir-los en servidors de l'entorn de producció.

Si una resposta lenta és pitjor que no donar cap resposta, llavors una resposta lenta i errònia o negativa és encara pitjor. Aquest patró proposa vigilar les fonts probables d'errors i avançar-se amb la resposta en cas que és pugui determinar que fallarà. Això no sempre es pot determinar, però si és el cas, no només estalvia temps de l'usuari sino recursos del sistema. Per tant abans de fer qualsevol crida, s'hauria de comprovar tot el que es pugui abans de fer-la. En primer lloc validar l'input i en segon comprovar, per exemple si el circuit breaker corresponent està tancat.

Les comunicacions són potencials fonts d'errors que s'han de tractar i protegir. La manera que proposa Michael Nygard és mitjançant el protocol de *handshaking*. Quan això no és possible, s'haurien de fer comprovacions d'estat: *Health-checks*, en cas que fer la comprovació sigui menys costosa que una crida que falla. També és recomanable utilitzar el handshaking per qualsevol protocol propi de baix nivell, per exemple a nivell de socket.

Test Harness representa un enfocament de desconfiança total amb respecte qualsevol dependència. Temps, format, contingut, mida de la resposta, o inclús el protocol de comunicació poden sortir del que s'havia especificat. Com tard o d'hora algun d'aquests problemes passaran s'ha d'estar preparat. Les proves del software ha d'incloure escenaris com els mencionats, i més.

Finalment un ben conegut patró de disseny: el baix acoblament, en aquest cas pel *middleware.* Aquell espai amb un desordre singular que permet la comunicació de sistemes que no s'havien dissenyat per treballar en conjunt. La Figura 5 mostra l'espectre d'acoblament pel middleware.

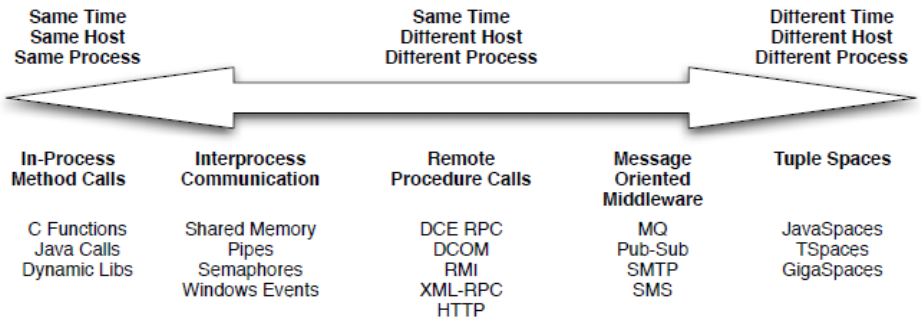


Figura 5. Els nivells d'acoblament i implementacions.

3.1.2 Capacitat

La capacitat d'un sistema és el rendiment(*troughput*) màxim sostenible pel sistema amb un temps acceptable de resposta per cada petició. La capacitat d'un sistema es defineix en funció de tres conceptes: velocitat per petició, rendiment en quant a numero de peticions processades per unitat de temps i l'escalabilitat. En aquest cas s'entén per escalabilitat incrementar la capacitat. Un greu problema que apareix en la anàlisis de la capacitat és la falta de linearitat. Per exemple, si un sistema pot donar suport a 10.000 usuaris utilitzant un 50% de la CPU, és fals deduir que el sistema hauria de suportar 20.000 en total.

Segons l'autor hi ha una sèrie de problemes o circumstancies que amenacen la capacitat d'un sistema. Aquestes són: Resource Pool Contention, AJAX Overkill,

Overstaying Sessions, Wasted Space in HTML, el botó de recarrega, Handcrafted SQL, Integration Point Latency.

3.1.2.1 Anti patrons

Quan el *pool* de threads actius que demanen accés a la base de dades supera el numero de connexions disponibles apareix el problema de Resource Pool Contention. El coll d'ampolla del sistema és la limitació del numero de recursos disponibles, ja que normalment les pool de connexions bloquegen indefinidament els threads en cas que no es puguin servir. Aquest és clarament un problema que amenaça la capacitat.

L'ús en excés de l'AJAX es pot convertir en un problema. Podria fer veure que el navegador esta penjat, l'increment de comunicació per la xarxa podria disparar-se i suposar una carrega massa gran i innecessària tan pel servidor com per l'aplicació.

La gestió del temps de caducitat de les sessions d'usuari és un altre factor important en qüestió de capacitat. Un fet curiós és que els usuaris que és recursos necessiten son els usuaris mes desitjats pel negoci. Però la memòria del servidor és escassa.

Sempre que no és te cura de la mida del HTML, aquest acaba sent una mica més gran. El problema apareix per dos raons. Primer l'increment innecessari de la memòria dels servidors web, i segon l'ús de l'ample de banda addicional inútilment. Un altre efecte que això provoca té com a actors els navegadors. Aquests, en funció de la mida del HTML mantenen una connexió durant més o menys temps.

El botó de recarrega del navegador està en mans de l'usuari, i això no són bones noticies. Cada cop que és prem el navegador abandona la petició anterior, obre un socket i fa una petició nova. El problema es que ningú mata la petició anterior, el servidor no sap que la pot descartar.

Combinar el servei d'ORMs amb consultes fetes a ma, encara que prometen eficiència són molt impredictibles. Tota la configuració de la base de dades esta preparada per les consultes dels ORM. L'autor dona diverses raons per no executar consultes fetes a ma, o minimitzar el seu ús.

Qualsevol comunicació remota comporta una certa latència, que normalment és 1000 vegades més gran que una crida local. Encara que aquest és un problema d'eficiència per l'usuari, pel sistema acabarà sent un problema de capacitat.

3.1.2.2 Patrons

Per evitar problemes de Resource Pool Contention s'han de configurar adequadament les Pool Connections. Aquestes poden a més d'evitar alentir tot el sistema, millorar la capacitat. A part, s'han de protegir tots els threads que demanin connexió a la base de dades.

Una bona implementació de cache pot reduir problemes de rendiment. Redueix la carrega del servidor de la base de dades. Però s'ha de verificar que aquest és el cas, s'ha de mesurar la tassa d'encerts i la freqüència d'ús dels contingut. En la configuració de la cache s'ha de tenir en compte el límit d'espai que pot ocupar la cache i implementar un bon mecanisme de flush.

En el mon web cada cop es requereix contingut dinàmic i específic. Portat a l'extrem arribem a trobar parts molt estàtiques a dins. Les parts estàtiques es poden pre-calcular.

Per últim, dins del tema de la capacitat, està el *Garbage Collector*. És la manera més rapida i fàcil per millorar la capacitat en aplicacions Java. Cal, doncs, analitzar el comportament del Garbage Collector, l'ús del *heap* i el temps que es triga per treure la

brossa, i ajustar la mida del heap. Configurar-lo bé porta avantatges de capacitat a més, té la capacitat de descobrir *memory leaks.*

Com a resum esquemàtic dels principis de resiliència que proposa el llibre tenim la Figura 3. Aquesta mostra les interaccions de patrons i anti-patrons. Els quadrats representen els patrons i els ovals els anti-patrons.

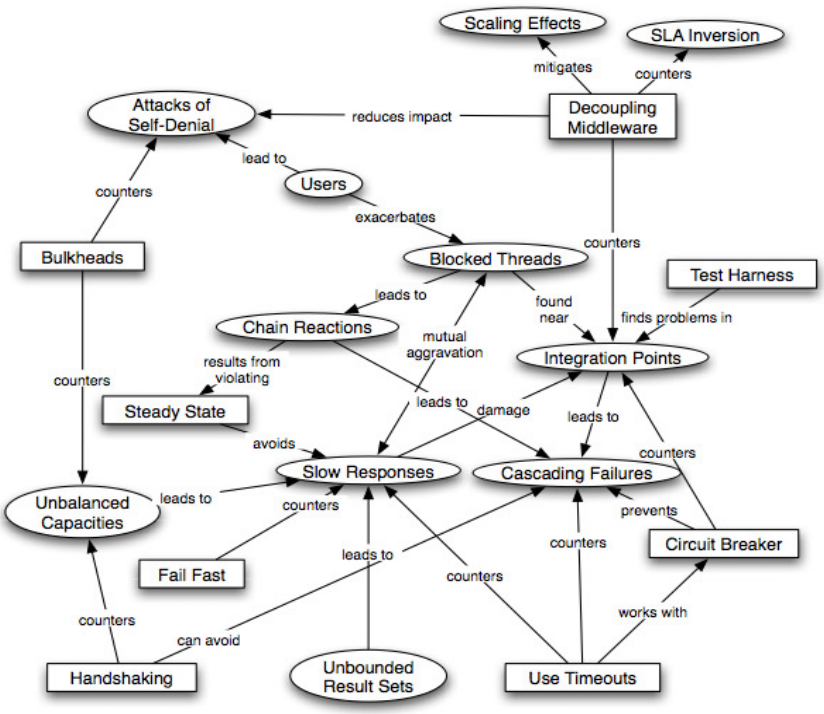


Figura 3. Interacció entre patrons i antipatrons

El llibre esta enfocat en trobar els anti-patrons i proposar patrons per solucionar la varietat d'errors que provoquen els primers.

3.2 Patterns of resilience

Uwe Friedrichsen té una sèrie de presentacions sobre la resiliència que són més recents[2][3]. En la primera, Patterns of resilience recupera molts dels principis que proposa Michael Nygard. També menciona i recomana el seu llibre, *Release it*. En quant als errors en sistemes complexos, parteix de tres hipòtesis. Els errors no són una excepció, no es poden predir i no es poden evitar. Per tant, recomana acceptar-los[8][2].

A continuació dona la seva definició de resiliència com la capacitat d'un sistema per manegar una situació inesperada. En el millor dels casos sense que l'usuari se n'adoni i en el pitjor cas amb una degradació del servei. Segueix el mateix enfocament a producció que trobem en el llibre, considerant la resiliència com una nova manera d'incrementar la disponibilitat del software. Després explica com els patrons de disseny influeixen sobre el sistema. Aquests s'expliquen fins a arribar a mostrar el codi. El resum visual dels patrons proposats per l'autor es pot veure a la Figura 14.

[8] "Do not try to avoid failures. Embrace them".

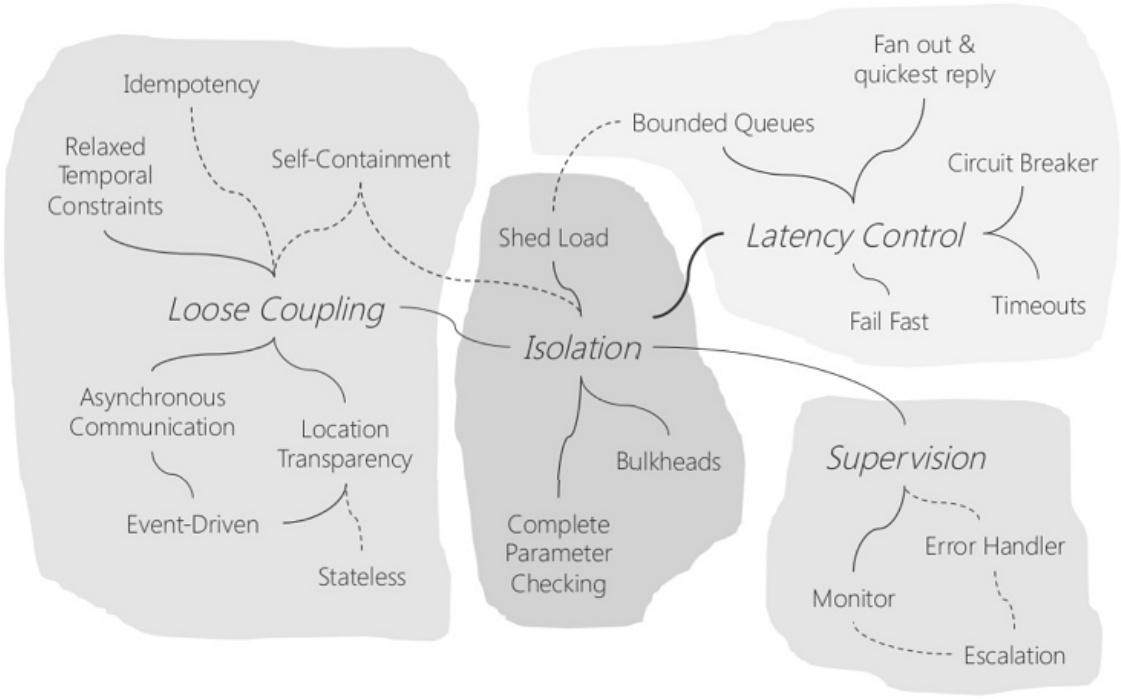


Figura 14. Patrons de resiliència i relacions.

Comentarem aquí un parell dels principis de resiliència que proposa en Uwe Friedrichsen. Dins del tema de l'aïllament, els Bulkheads ja els trobem en el llibre d'en Michael Nygard. Per tant els dos que queden són: Complete Parameter Checking i Shed Load.

3.2.1 Complete Parameter Checking

Per argumentar la comprovació dels paràmetres de tots els mètodes, invoca la llei de Postel[5]. Aquesta llei sosté que en el disseny de software un ha de ser liberal en el que accepta però conservador en el que envia. Es recomana, doncs que cada

paràmetre tingui un tipus de dades específic. Així és protegeix el mètode de crides malicioses.

3.2.2 Shed Load

Aquest principi és preocupa de guardar els recursos. Per evitar una sobrecarrega de peticions, s'ha de posar un *porter*[9] davant dels recursos. Per tant és limita la quantitat de peticions en funció de la carrega que el sistema ja en té.

[9] Gatekeeper

3.3 Resilience reloaded

En la segona presentació[3] a més de proposar encara més principis de resiliència, ja es proposa una arquitectura per les aplicacions resilient. Es fa, doncs, distinció entre els principis de resiliència que caracteritzen el software (Core), i els que han de ser externs(Detecció, Tractament i Prevenció). Tal com es pot veure a la Figura 21, ja la resiliència inclou prevenció d'errors.

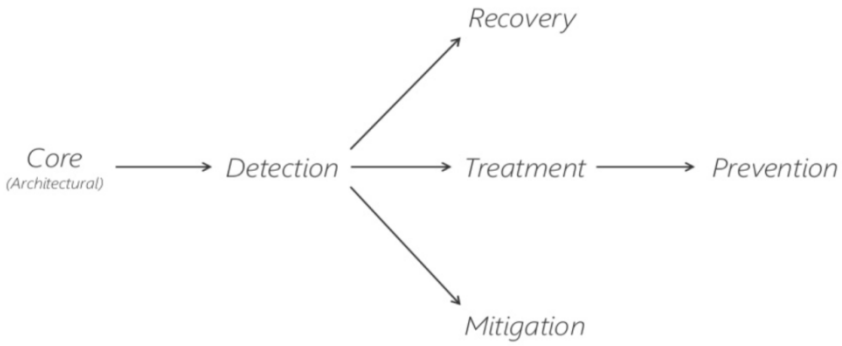


Figura 21. Arquitectura per a software resilient.

3.4 A Framework for Self-Healing Software Systems

En aquest article, resultat del seu doctorat, Nicolò Perino proposa un framework per aconseguir resiliència software. Com ja hem mencionat més a munt, la idea consisteix a aprofitar la redundància a nivell de mètode de les llibreries. Segons el seu anàlisis hi ha una alta redundància en les llibreries Java. A la Taula 3 podem veure el resultats numèrics.

Library	Guava	SWT	JodaTime
Classes studied	116	252	12
Number of Equivalence Sequences	1715	1494	135
Average per class	14.78	5.93	11.25

Taula 3. Redundància de tres llibreries Java.

El framework introdueix, degut al cost de les instruccions *try-catch,* un overhead entre 9% -140%. Els resultats de les proves amb algunes aplicacions mostra un percentatge d'auto recuperació entre el 20% i el 50%

3.5 Principis proposats

En el cas ideal d'aplicació resilient des del nostre punt de vista hauria de complir les següents característiques. Des de la fase de disseny ja és comenci a plantejar els principis de resiliència aplicables. La implementació de la resiliència està totalment desacoblada de l'aplicació. Els principis de resiliència estan desacoblats entre ells. De tal manera el nivell de resiliència es pot incrementar o disminuir amb facilitat. Entenem però que, des d'un enfocament determinista, hi ha principis de resiliència que no arriben a complir aquest ideal. Inclús entre els principis, que proposem i hem implementat nosaltres, es troben alguns que no compleixen aquest ideal.

Hem començat tractant el tema de la connectivitat. Cada cop hi ha més velocitat a les xarxes de comunicació. Cada cop es fan més aplicacions que utilitzin les

dades. Ens estem apropant a la era IOT. Però qualsevol xarxa encara esta lluny de ser infal·lible. Per tant, pensem que s'ha d'explorar al màxim les funcionalitats que podria donar una aplicació a l'usuari encara que temporalment l'usuari es trobi sense connexió.

El principi de resiliència que hem proposat l'anomenem mode offline. Algunes aplicacions ja l'implementen i consisteix en utilitzar el concepte de cache persistent en el dispositiu de l'usuari. Així les dades que pot produir l'usuari i que s'han de transmetre al servidor poden guardar-se mentre no hi hagi connexió. L'aplicació és la encarregada de comprovar si la connexió s'ha establert. Aquesta també tindrà l'usuari informat en qualsevol canvi en la disponibilitat de les funcionalitats afectades.

La comunicació és veu afectada per un tall de connexió en els dos sentits. L'usuari no pot ni rebre ni enviar. Tant la memòria com capacitat de processament dels dispositius mòbils han anat avançant. Seguint la llei de Moore han arribat a capacitats de supercomputadors d'altres èpoques, veure Figura 2. Aprofitant aquesta capacitat podem implementar un concepte ben conegut, la cache, per intentar aprimar el vuit de funcionalitats que provoca el fet d'estar offline.

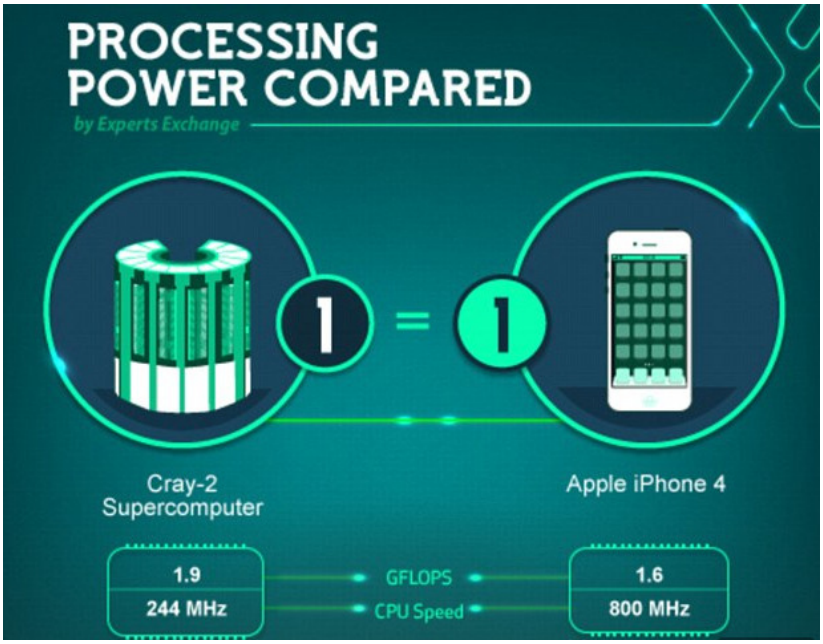


Figura 2. Cray-2 1986 i iPhone 4 2010

3.6 Comentaris finals

Moltes de les propostes que s'han fet, començant pel llibre, estan enfocats a una resiliència que s'aconsegueix en la fase del disseny. Estem d'acord que la resiliència comença en aquesta fase, però no es queda allà. També insistim en una solució desacoblada i incremental. Considerem que el pensament empàtic cap a l'usuari en qualsevol fase és la forma més adequada de generar principis de resiliència. Amb usuari, no ens referim simplement a l'usuari final, el mateix desenvolupador és un usuari de l'aplicació.

En el cas del treball d'en Nicolò Perino trobem alguns problemes en el plantejament. Per una banda, l'anàlisi de les llibreries, en cerca de la redundància és fa de manera manual. Per una altra banda, la investigació s'ha fet segons al *Javadoc*. Trobem insuficient raó per considerar dos mètodes redundants només perquè el Javadoc ho afirmi.

Tal com havíem explicat, la nostra visió sobre aplicacions resilient és d'una o més capes que es poden afegir. No tots els principis de resiliència requereixen el mateix nivell d'intrusisme, per tant, és possible afegir resiliència a posteriori al software.

4 Execució del projecte

4.1 Aplicació

Per la implementació dels principis de resiliència hem escollit aprofitar una aplicació que és el resultat de l'assignatura de PES anomenada Hangaround. L'aplicació té coma objectiu facilitar l'accés a vida social de les persones amb discapacitats motrius. El cas emblemàtic d'aquest públic objectiu són les persones que utilitzen cadires de rodes. Amb l'ajuda de la comunitat d'usuaris que la fan servir, l'aplicació proporciona informació sobre el nivell d'adaptació d'espais o locals d'accés públic. A la Figura 1 es pot veure que l'aplicació segueix el paradigma de client-servidor i les seves dependències, tant la part client com de la part servidor.

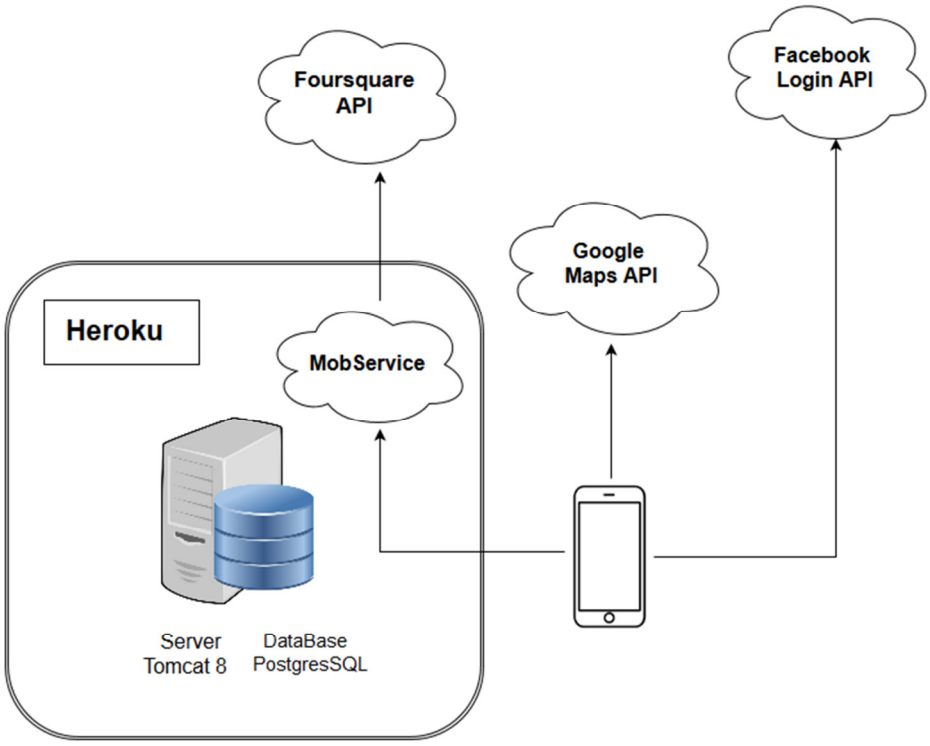


Figura 1. Infraestructura i dependències.

4.1.1 Servidor: MobService

La part servidor és un servei web amb arquitectura API REST, implementada en Java, corre sobre un Tomcat i esta allotjada en Heroku. S'encarrega principalment de la persistència de les dades. Té com a dependència principal l'API de Foursquare que utilitza per a proveir llocs al voltant de la ubicació o direcció cercada. En els casos d'ús només farem servir cerques de ciutats. Foursquare proporciona els llocs: locals, bars, museus, etc. Aquests llocs es guarden en una base de dades relacional en el servidor.

La cerca de llocs es fa en Foursquare però des del la nostra API. Els paràmetres que utilitzarem per fer la cerca són: *near* i *limit*. Amb el primer especifiquem una ciutat i amb el segon limitem en numero de resultats que desitgem mostrar. En els casos d'ús variarem el seu valor entre 1 i 25 segons convingui. Per a cada lloc que no és troba ja en el backend és guarda, amb un thread en background, amb un nivell d'adaptabilitat desconegut: UNKNOWN.

Els usuaris poden fer valoracions en relació amb el nivell d'adaptabilitat dels llocs. Hi ha quatre nivells: UNKNOWN – desconegut, UNADAPTED – sense adaptar, PARTIAL – parcial, i TOTAL. La part servidor determina el nivell d'adaptabilitat d'un lloc en funció de les valoracions que han fet els usuaris sobre aquell lloc. Una valoració

requereix tres paràmetres: accés, serveis(wc) i ascensor. Un lloc té com a nivell d'adaptabilitat la que correspon a la ultima valoració que li han fet.

El primer es refereix a l'accés i mobilitat dins del perímetre del lloc, principalment rampa a l'entrada i amplitud dels passadissos. El segon comprova l'existència de serveis adaptats. Tant el primer com el segon tenen una codificació binaria de cert o fals. Finalment la presència o absència de l'ascensor és el tercer paràmetre. Aquest només tindria sentit avaluar-lo en el cas en què el local té més d'una planta. La codificació d'aquest paràmetre consisteix en un enumerable amb els següents valors: HAS – el lloc disposa d'ascensor destinat a l'ús públic, NO_NEED – el lloc només té una planta, finalment HAS_NOT – el lloc té més d'una planta però no disposa d'ascensor.

4.1.2 Client: Hangaround

En la part client disposem d'una aplicació mòbil per a dispositius android. Esta implementada en Java, aplicació nativa. Com a principals dependències té l'API de Facebook per facilitar l'accés dels usuaris sense la necessitat de crear un compte nou. Els llocs que proporciona Foursquare es representen en un mapa. Per mostrar els llocs s'utilitza l'API de Google Maps.

Després de fer la cerca de llocs com s'explica a dalt, per a cada lloc recuperat de Foursquare es busca el nivell d'adaptabilitat que té en el backend. Actualitzant un per un els llocs mostrats, amb marcadors de l'API de Google Maps, en el mapa. Aquests marcadors mostren el nivell d'adaptabilitat dels llocs que representen segons la llegenda de colors que es pot veure a la Taula 2.

Color	Descripció
GREEN	Totalment adaptat
YELLOW	Parcialment adaptat
RED	Cap adaptació
CYAN	Desconegut (per defecte)

Taula 2. Codi de colors dels marcadors en el mapa.

Clicant els marcadors es pot fer una valoració sobre aquell lloc mitjançant el formulari que apareix mitjançant un pop-up, com es pot veure a la Figura 3.

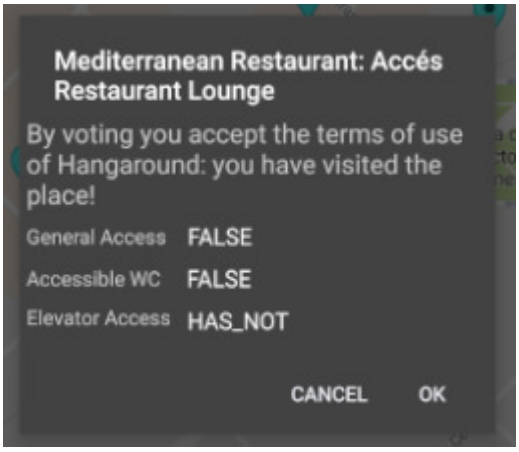


Figura 3. Pop-up amb formulari per fer una votació.

4.2 Casos d'ús

Els principis de resiliència que hem implementat són els següents, mode offline, recuperació d'excepcions NullPointerException i vigilància a nivell de mètode mitjançant un mòdul d'intel·ligència artificial.

4.2.1 Mode offline

Com ja hem mencionat(link als principis), la motivació d'aquest principi és donar el màxim de funcionalitats en cas de desconnexió. Per una banda prova de mitigar els efectes de perdre la connexió al backend. Per l'altre banda la pèrdua de connexió del mòbil a internet. La capa de resiliència s'encarregarà d'abordar els dos problemes.

4.2.1.1 Pèrdua de connexió al backend

La xarxa pot fallar o l'aplicació que s'executa en el backend pot fallar. Per un error propi o per culpa d'una dependència. Siguin quines siguin les causes o els causants d'aquests errors, el backend pot arribar a no ser disponible[10]. Tal com ha estat dissenyada i implementada l'aplicació no és capaç d'oferir cap funcionalitat. Qualsevol intent de connexió al backend retorna un missatge d'error. En termes de

[10] El backend perd la disponibilitat si dona una resposta errònia o dona una resposta massa tard.

disponibilitat, la indisponiblitat del backend provoca la indisponiblitat de totes les funcionalitats de l'aplicació.

Com ja havíem dit, vam implementar una cache per a cada una de les dues funcionalitats bàsiques: consultar llocs i el seu nivell d'adaptabilitat i valorar llocs. La cache, en els dos casos, és persistent i es troba en el dispositiu; no depenem de la xarxa i per tant és prou ràpida. Les cache són persistents, en una base de dades NoSQL (key-value) proporcionada per l'Android SDK, SharedPreferences.

En primer lloc hem implementat una cache per a les cerques; cerca-resultat. On la cerca és el text cercat en format String i el resultat és el conjunt de llocs que torna el backend per aquella cerca en format JSONArray. La lectura de la cache no es fa indefinidament. L'usuari esta informat que s'ha perdut la connexió amb el servidor de l'aplicació. Però no només això sino que en background s'inicia un thread que cada deu segons va comprovant si s'ha restablert la connexió amb el servidor. En tornar a estar disponible el servidor, l'usuari torna a estar notificat i la funcionalitat torna a estar al cent per cent disponible. El mòdul de resiliència ha provocat que l'aplicació toleri la fallida del backend i que torni a l'estat òptim tan aviat com sigui possible.

En segon lloc hem implementat una cache per a les valoracions. Encara que el backend no estigui disponible, l'usuari pot interaccionar amb l'aplicació valorant llocs. La cache guardarà totes les valoracions efectuades per part de l'usuari mentre està sense connexió amb el servidor. Quan es detecti que el servidor torna a estar disponible, entrada per entrada s'envien les valoracions de la cache per guardar-les en el backend.

4.2.1.2 Pèrdua de connexió a internet

El segon problema que havíem mencionant i que te a veure amb la connexió és el fet que el dispositiu es trobi en una zona de cobertura insuficient. Tot i ser semblant al primer hi ha una problemàtica afegida ja que l'aplicació té més dependències no nomes amb el backend. Per una banda el login mitjançant l'API de Facebook i el mapa per ubicar els llocs proporcionada per l'API de Google Maps. Degut als termes d'ús d'aquesta darrera API, Captura 4, hem hagut de canviar la representació dels llocs quan el dispositiu no esta connectat. Només està autoritzat fer ús de cache dins del servei que Google proporciona.

Per no canviar tota la implementació de l'aplicació original s'ha decidit fer una vista en format llista amb els llocs, nova disponible a la Captura 5. Utilitzant l'API d'OpenStreetMap no s'hagués hagut de canviar la representació; menys d'intrusió.

10.5 Intellectual Property Restrictions.

a. No distribution or sale except as permitted under the Terms. You will not distribute, sell, or otherwise make any part of the Service available to third parties except as permitted by these Terms.

b. No derivative works. You will not modify or create a derivative work based on any Content unless expressly permitted to do so under these Terms. For example, the following are prohibited: (i) creating server-side modification of map tiles; (ii) stitching multiple static map images together to display a map that is larger than permitted in the Maps APIs Documentation; or (iii) tracing or copying the copyrightable elements of Google's maps or building outlines and creating a new work, such as a new mapping or navigation dataset.

c. No use of Content outside the Service. You will not use any Content outside of the Service except as expressly permitted to do so in Subsection (d). For example, you will not export or save the Content to a third party's platform or service.

d. No caching or storage. You will not pre-fetch, cache, index, or store any Content to be used outside the Service, except that you may store limited amounts of Content solely for the purpose of improving the performance of your Maps API Implementation due to network latency (and not for the purpose of preventing Google from accurately tracking usage), and only if such storage:

i. is temporary (and in no event more than 30 calendar days);

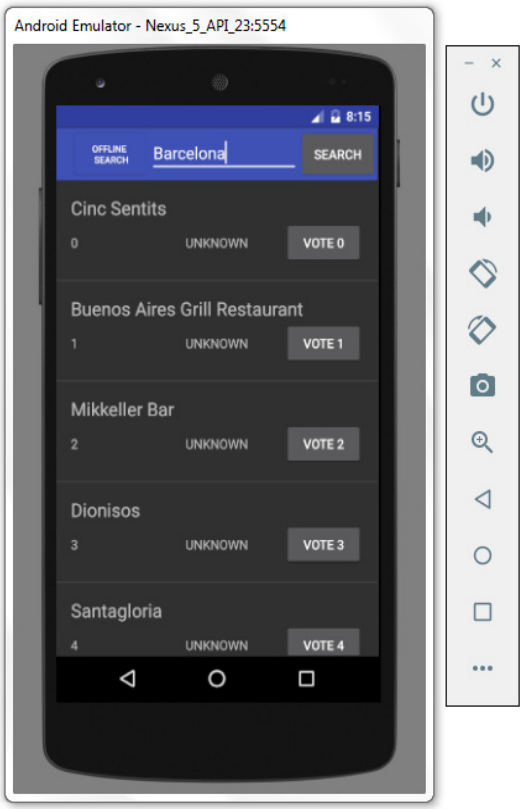
ii. is secure;

iii. does not manipulate or aggregate any part of the Content or Service; and

iv. does not modify attribution in any way.

e. No mass downloading. You will not use the Service in a manner that gives you or a third party access to mass downloads or bulk feeds of any Content. For example, you are not permitted to offer a batch geocoding service that uses Content contained in the Maps API(s).

Captura 4. Extret dels termes d'ús de Google Maps API



Captura 5. Nova vista per mostrar els llocs en mode offline.

Hi ha una limitació del model escollit deguda a no fer pre-fetch i per tant els llocs disponibles per a consultar sense cap impediment són només els llocs prèviament cercats. L'usuari pot trobar-se utilitzant les dades i encara sent potents l'autonomia dels dispositius mòbils està condicionada per la capacitat de la bateria. Tenint en compte aquests dos factors hem considerat que el millor enfocament seria no consumir ni dades ni espai en el dispositiu per fer pre-fetch.

4.2.2 Errors interns: NullPointerException

Els errors s'amaguen en el codi de qualsevol projecte de software. Passem, doncs, de les dependències d'una aplicació als errors inesperats que pot produir la mateixa aplicació. Degut a que alguns errors no s'arriben a trobar en la fase de test es solen capturar tots els errors i crear vistes personalitzades d'una varietat d'errors. Hi ha varies raons per actuar així, com ara la seguretat o inclús la confiança de l'usuari. Però aquí volem aplicar un segon principi de resiliència. No esperem que l'aplicació simplement informi de l'error sino que provi d'arreglar-lo. Aquest principi l'aplicarem a la part servidor de l'aplicació.

Suposant que l'ús que li dona un usuari encara i sent correcte provoca una excepció, com per exemple: NullPointerException. El comportament que desitgem que l'aplicació tingui en aquest cas és de provar de recuperar-se d'aquest error i acabar servint l'usuari amb la resposta desitjada. Les limitacions temporals han impedit la

construcció d'un agent Java que faci la substitució de classes en calent. En conseqüència hem emprat un plugin ja existent, jRebel.

jRebel és capaç fer la substitució en calent, sense que calgui reiniciar el servidor, sense haver d'interrompre[11] la interacció de l'usuari amb l'aplicació. El seu funcionament consisteix en vigilar per una banda l'aplicació desplegada i per l'altre el directori target. En detectar que el timestamp és diferent executa la substitució. Això ens porta al segon ingredient necessari per que aquest principi es pugui aplicar. El mòdul de resiliència se li ha d'especificar la ruta al repositori de classes (l'equivalent al directori target en l'ús convencional del plugin). Aquest repositori de classes compilades podria estar estructurat de manera que proveeixi diverses versions.

En aquest exemple, de caire didàctic, no hem tingut en compte aquesta possibilitat. Per simplicitat també, per l'execució de la demostració el plugin s'ha configurat per un Tomcat que corre en local. L'error s'ha introduït en la primera versió d'una classe FourSquareController i sense altres dependències[12]. Però en un cas real s'hauria de buscar la classe i les seves classes dependents, ja que s'haurien de substituir totes en conjunt. Aquestes podrien estar agrupades en *minijars* per facilitar l'aplicació d'aquest principi. Per tant, recomanem aplicar aquest principi en arquitectures de microserveis.

[11] En aquest cas l'usuari només pateix una demora de 2 segons en rebre la resposta.
[12] L'error introduït afecta una única classe per tant desapareix en fer una simple substitució de la classe.

Un aspecte important aquí seria s'hauria de decidir si s'aplica la substitució en temps real, però amb el risc que la solució torni a donar algun altre error, o executar algunes proves abans de fer la substitució.

Considerem com a cas ideal d'aplicar resiliència a una aplicació mitjançant un disseny extensible, es a dir, una arquitectura de plugins. Perseguim un nivell mínim d'intrusisme, de tal manera que els principis de resiliència que es considerin es vagin afegint a l'aplicació sense estar acoblats. En aquest sentit hem implementat els primers dos principis de resiliència amb el paradigma de programació orientada als aspectes(AOP). Degut a limitacions imposades per Android, ha calgut implementar de manera acoblada els mètodes que permeten comprovar en background si s'ha establert la connexió amb el servidor o si el dispositiu torna a tenir connexió a internet.

Per raons de temps i costos s'han pres algunes decisions en quant a la implementació. En projectes reals l'experiència dels desenvolupadors i les necessitats especifiques de cada aplicació donen peu a explorar nous principis de resiliència.

5 Conclusions

Les conclusions del treball. En primer lloc hem d'avaluar el compliment dels objectius, després avaluar la planificació – teoria vs pràctica,

5.1 Treballs futurs

La tendència del software d'incrementar el nivell de complexitat i distribució fa cada cop més necessari un enfocament resilient. Encara que des del punt de vista determinista encara queden moltes coses per fer veiem que el Machine Learning seria l'enfocament més adequat per dotar el software de resiliència. Per tant, com a possible projecte de futur seria cercar o aplicar principis de resiliència mitjançant xarxes neuronals. Aquestes tenen la capacitat no només de detectar els errors. Per exemple, si es determina que un mètode ja no retorna el que hauria, podrien prendre el control, i proporcionar la sortida en base a l'entrada.

Una altre concepte explotable en aquest sentit surt del funcionament del nostre cervell. En el seu llibre, I el cervell va crear l'home, Antonio R. Damasio sosté que el cervell té mapejat el cos humà.

6 Annex

6.1 Diagrames de Gantt

	Name	Duration	Start	Finish
1	Part I	14.9w	04/15/2016	07/28/2016
2	Fer recerca	3w	04/15/2016	05/05/2016
3	Llegir i estudiar	3w	05/05/2016	05/25/2016
4	Resumir i criticar	3w	05/25/2016	06/14/2016
5	Proposar	4w	06/14/2016	07/11/2016
6	Part II	15w	09/16/2016	12/29/2016
7	Requisits	2w	09/16/2016	09/29/2016
8	Disseny	2w	09/29/2016	10/12/2016
9	Implementació	7w	10/12/2016	11/29/2016
10	Verificació	3w	11/29/2016	12/19/2016

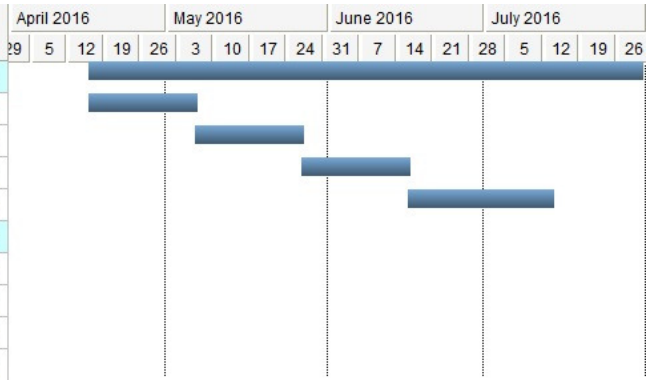


Diagrama 1. Tasques de la primera part en diagrama de Gantt.

	Name	Duration	Start	Finish
1	Part I	14.9w	04/15/2016	07/28/2016
2	Fer recerca	3w	04/15/2016	05/05/2016
3	Llegir i estudiar	3w	05/05/2016	05/25/2016
4	Resumir i criticar	3w	05/25/2016	06/14/2016
5	Proposar	4w	06/14/2016	07/11/2016
6	Part II	15w	09/16/2016	12/29/2016
7	Requisits	2w	09/16/2016	09/29/2016
8	Disseny	2w	09/29/2016	10/12/2016
9	Implementació	7w	10/12/2016	11/29/2016
10	Verificació	3w	11/29/2016	12/19/2016

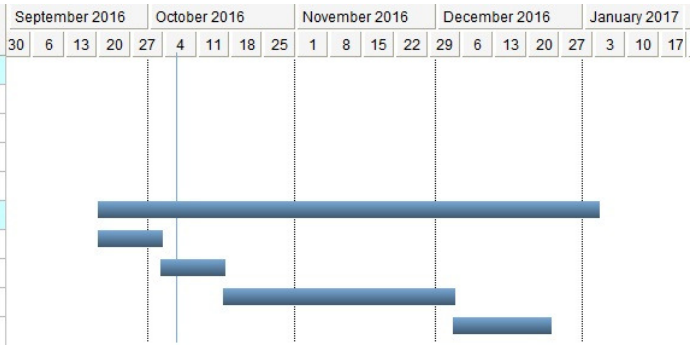


Diagrama 2. Tasques de la segona part en diagrama de Gantt

6.2 Glossari

Resiliència : Capacitat de l'individu per a afrontar amb èxit una situació desfavorable o de risc, i per a recuperar-se, adaptar-se i desenvolupar-se positivament davant les circumstàncies adverses.

Aplicació resilient :

7 Bibliografia

[1] B. Jonas, “Without Resilience Nothing Els Matters,” 2015, p. 193.

[2] Friedrichsen Uwe, “Patterns of resilience,” 2015. [Online]. Available: http://www.slideshare.net/ufried/patterns-of-resilience.

[3] Friedrichsen Uwe, “Resilience reloaded - more resilience patterns,” 2016, p. 102.

[4] M. T. Nygard, *Release It! Design and Deploy Production-Ready Software*. 2007.

[5] P. Jon, “RFC 761,” p. 83, 1980.