M.DESCHAMPS Mathieu M.BIGNAUX Ronan

Tech-stories : Aspects des projets FLOSS R&D.

Table des matières

Le processus R&D

R&D FLOSS

Introduction

Cartographie

Marché : plus 4 Milliards €

Acteurs : SSII & éditeurs de logiciels. Presque 50% existent depuis plus de 15 ans

Secteur : environ 70% pour la défense, le militaire, l'espace et l'automobile

CA : 90% réalisé en France.

Invest. R&D embarquée : 7% du CA

Activité embarqué minoritaire : pour environ 80% des Acteurs

Modèle économique : 44% Mixtes (4% de "puristes")

Pérennisation technologie embarqué : 70% privateur (14% OpenSource)

57% Source Closing

13% Matériels IP ""standard""

Commentaires

Sources

Prév. du marché Syntec Ingénierie 2008

Livre Blanc des premières assises françaises du logiciel embarqué "logiciel"

Qu'est-ce que la Recherche Développement ?

"Systematic activity combining both basic and applied research, and aimed at discovering solutions to problems or creating new goods and knowledge."

Un processus au long cours composé d'activités combinant recherche fondamentale et opérationnelle dans le but de concevoir, de réaliser et d'améliorer un produit/service au moyen d'une solution innovante.

Commentaires

Source

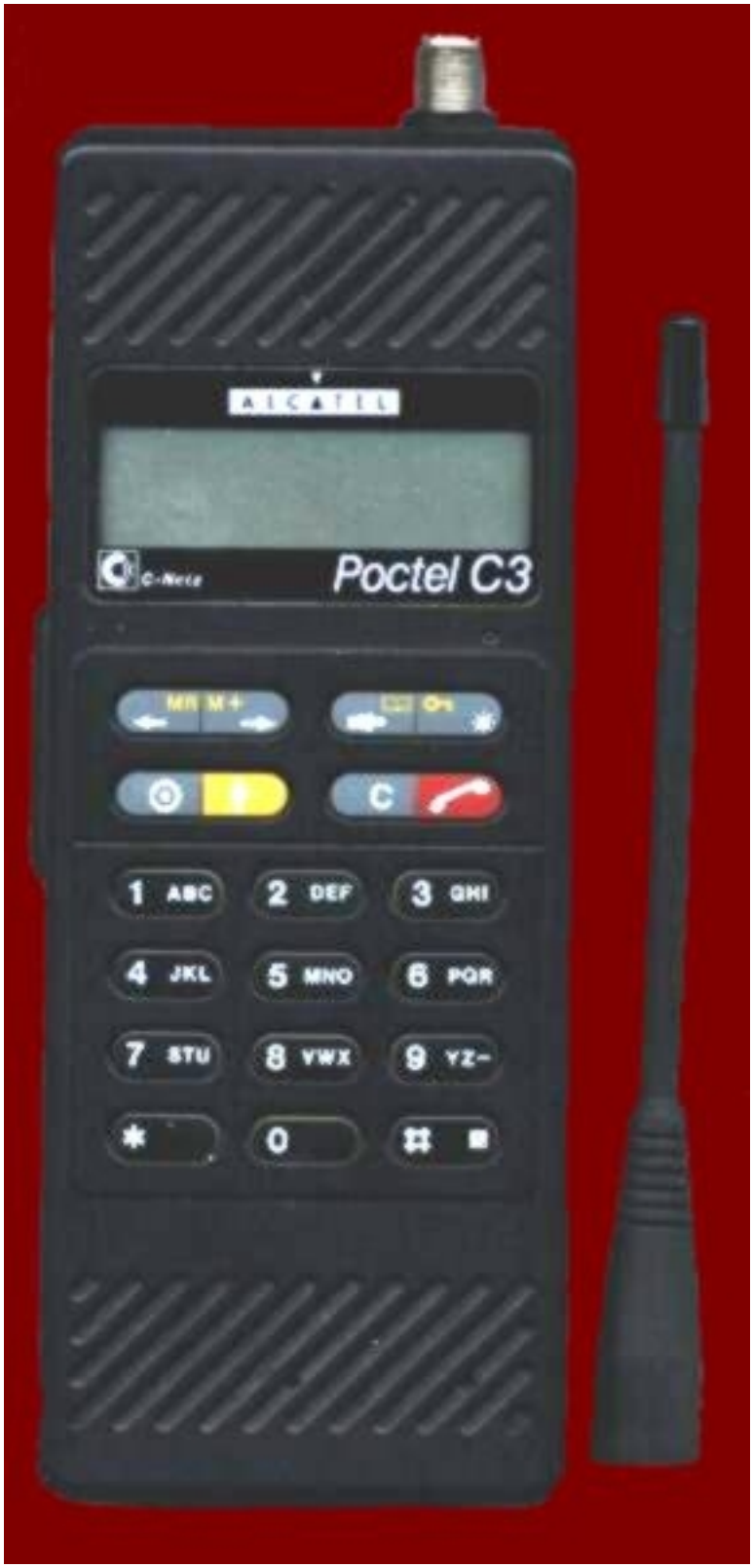
www.businessdictionary.com

20 ans de RD Télécom : le téléphone portable Alcatel

S'il est né au début du siècle dernier, le téléphone n'est devenu (trans)portable qu'au

Alcatel SEM 335 M/TC

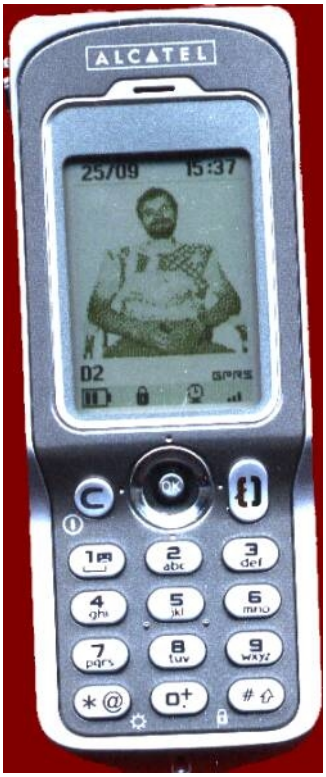
début des années 1990.


ALCATEL
C-Netz
Poctel C3
MR M+
C
1 ABC
2 DEF
3 GHI
4 JKL
5 MNO
6 PQR
7 STU
8 VWX
9 YZ-
*
0
#

Poctel C3

Alcatel One Touch Easy

Alcatel One Touch Easy Bi-Bande



Alcatel One Touch Easy 715

OV T770

Crédits Photos : Copyright 2008 - Handy-Sammler.de

Commentaires

L'Alcatel SEM 335 M/TC

1988

3,77kg

taille d'une valise

Poctel C3

1990

585g

35 cm x 8 cm x 3,5 cm

OTE

1997

200g

15 cm x 6 cm x 4 cm

(passage à la carte à puce SIM)

Alcatel One Touch Easy Bi-Bande

1999

200g

15 cm x 5 cm x 3cm

Dualband 900/1800MHz

OTE 715

2002

88g

11 cm x 4 cm x 2 cm

Alcatel OT-V770

2008

75g

10 cm x 5 cm x 1 cm

Crédits Photos (danke Elmi)

Pourquoi démarrer une activité R&D ?

Objectifs

Qualifier/Quantifier une faisabilité : R&D "exploratoire", la recherche d'opportunités nouvelles

Maintenir une position sur le marché : R&D "exploitation", la recherche depuis l'existant sur objectifs

Valider une démarche, certifier un procédé : R&D "consolidation", la recherche servant de R&D étalon

Commentaires

Google (allocation de temps "dev libre" pour les projets persos des developpeurs)

Alcatel

Durée : une précision.

Une R&D ne dure pas le temps de remplir un objectif...

elle dure le temps de l'épuiser.

Et à l'échéance on estime si l'objectif est rempli.

Commentaires

Remplir l'objectif

Il faudrait sortir un produit immédiatement sur le marché, attendre la commercialisation, les ventes, et leur analyse pour effectivement

indiquer objectivement de l'accomplissement. D'où la notion de "time to market" qu'il faudrait toujours réduire.

Plus time to market court, plus rapide et le retour du marché. Donc l'intérêt de poursuivre la R&D.

Qu'est-ce que la R&D des Logiciels Libres et Open Sources (FLOSS) ?

La même chose avec les bons logiciels ? ;)

Pas seulement...

- Les outils,
- Les procédés,
- Les acteurs,
- L'organisation de projet

... diffèrent.

La R&D FLOSS s'épanouit grâce à l'effort conjoint des Logiciels Libres et Open Sources sur deux fronts dans :

- les projets fondateurs pour l'ouverture et
- les entreprises commerciales pour l'extension

Commentaires

Effort spécifique sur deux fronts FLOSS épanouit la R&D FLOSS

- les projets fondateurs pour l'ouverture (fondation, spécification, développement, normalisation) ;
- les entreprises commerciales pour l'extension (diffusion, exploitation, développement, structuration).

Contribution Kernel CAN avec Volkswagen

Deux années de RD FLOSS sur le projet OpenMoko

FIC Neo 1973

La R&D liée à OpenMoko donne naissance un produit innovant exploité par la société FIC.

Commentaires

Un intérêt pour la liberté... matérielle

OpenMoko ouvre également les schématiques CAD du PCB et les mets à disposition.

Pas si fréquent dans le métier.

Prix : environ 300$ à 450$ (190€ à 290 €)

Pour conclure

C'est un continuum qui se découpe en phases, en projets indépendant (en principe !) du cycle de vie d'un produit.

- But : donner naissance à un produit/service issu de la plus-value apportée à l'état de l'art.
- Constat : c'est l'effort commun des initiatives privées et publiques qui permet son essor.

La R&D FLOSS a donc un caractère essentiel et cyclique, synergique et indépendant.

Commentaires

Appui sur le rapport RD/produit.

Parfois, la RD sur une phase ne conduit systématiquement à un produit/service innovant vendu sur le marché.

Mais une RD sur un projet s'arrêtera si son produit meure.

Organisation

Fonctionnement

Constats

L'utopie des spécifications immuables perdure.

L'erreur du Test-driven développement

Si les DCU des méthodes UP sont presque inefficace pour la méthodologie en "couche basses" ...

Ils restent applicables pour l'organisationnel :

- Qui est celui qui va utiliser mon code/mon service ?
- Comment va-t-il l'utiliser ? Quels sont ses entrées ?
- etc..

Les méthodes agiles gagnent du terrain...

Commentaires

Utopie

Spécif immuables :

Soyez en sur votre client changera d'opinion, votre chef de projet aussi alors vous certainement.

Alors où est le problème ? Lorsque le modèle fige à une décision qui faut néanmoins revoir.

Il est encore pas rare de se voir dire : "il faudra utiliser la lib Zulu" avant de commencer le cahier des charges

Test-driven :

Comment tester les tests ? ;)

Pas tjs de board d'éval. Les émulateurs ne parent pas au pb tester les periph ce qui revient à dev. des drivers virtuel pour l'emulateur.

DCU : Diagramme de cas d'utilisation

Le DCU doit donc s'étendre à l'environnement de projet et nos exclusivement au SI.

Constat : ces questions issus des DCU admettrons des réponses qui varieront au cours du projet

-> Accepter de se remettre en question / Accepter les outils qui

l'encourage. (courage)

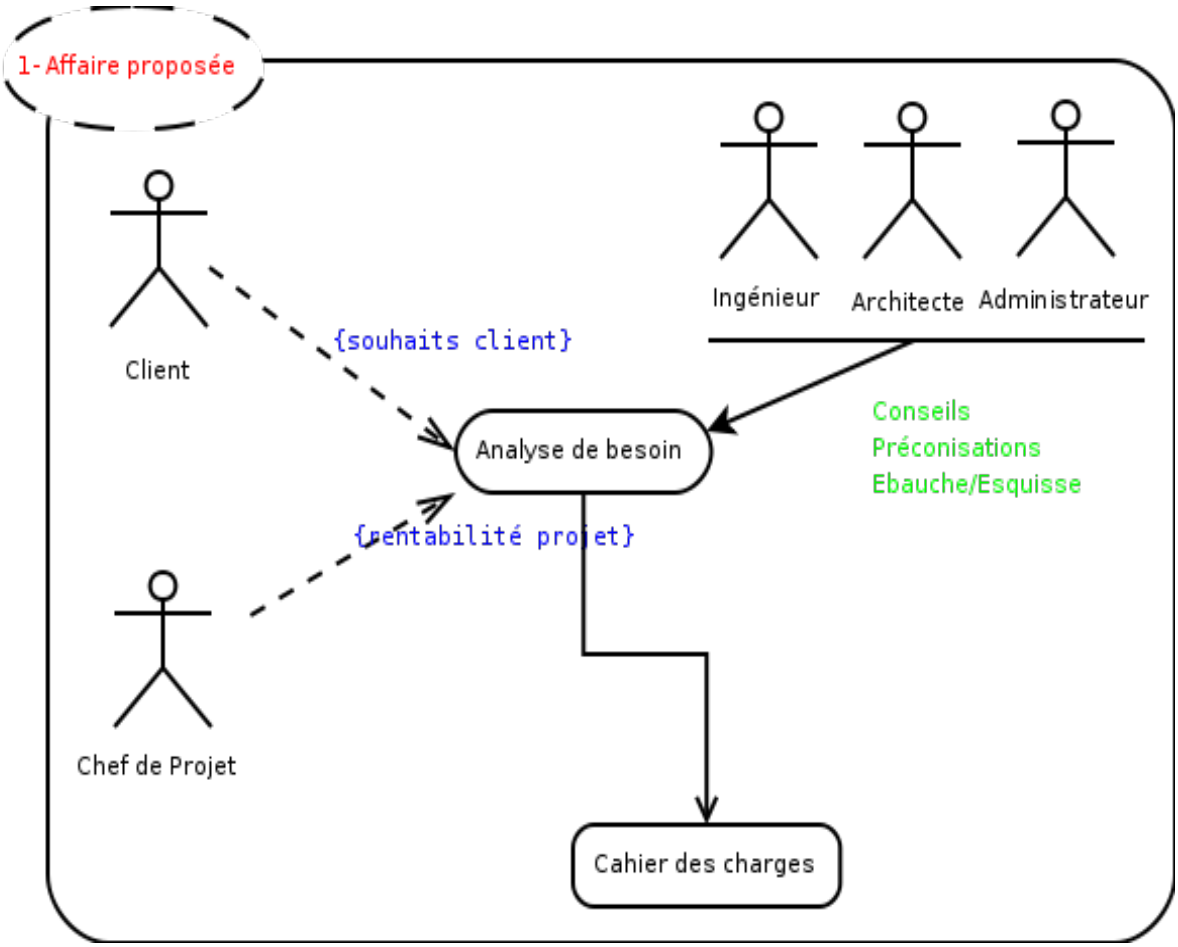
méthodes agiles , instinctives et de bon sens, ont de bons aspects salutaires !

mais Attention à ne pas plaquer par réflexe de "l'agilité" partout, Fameux fantasme Pavlovien :

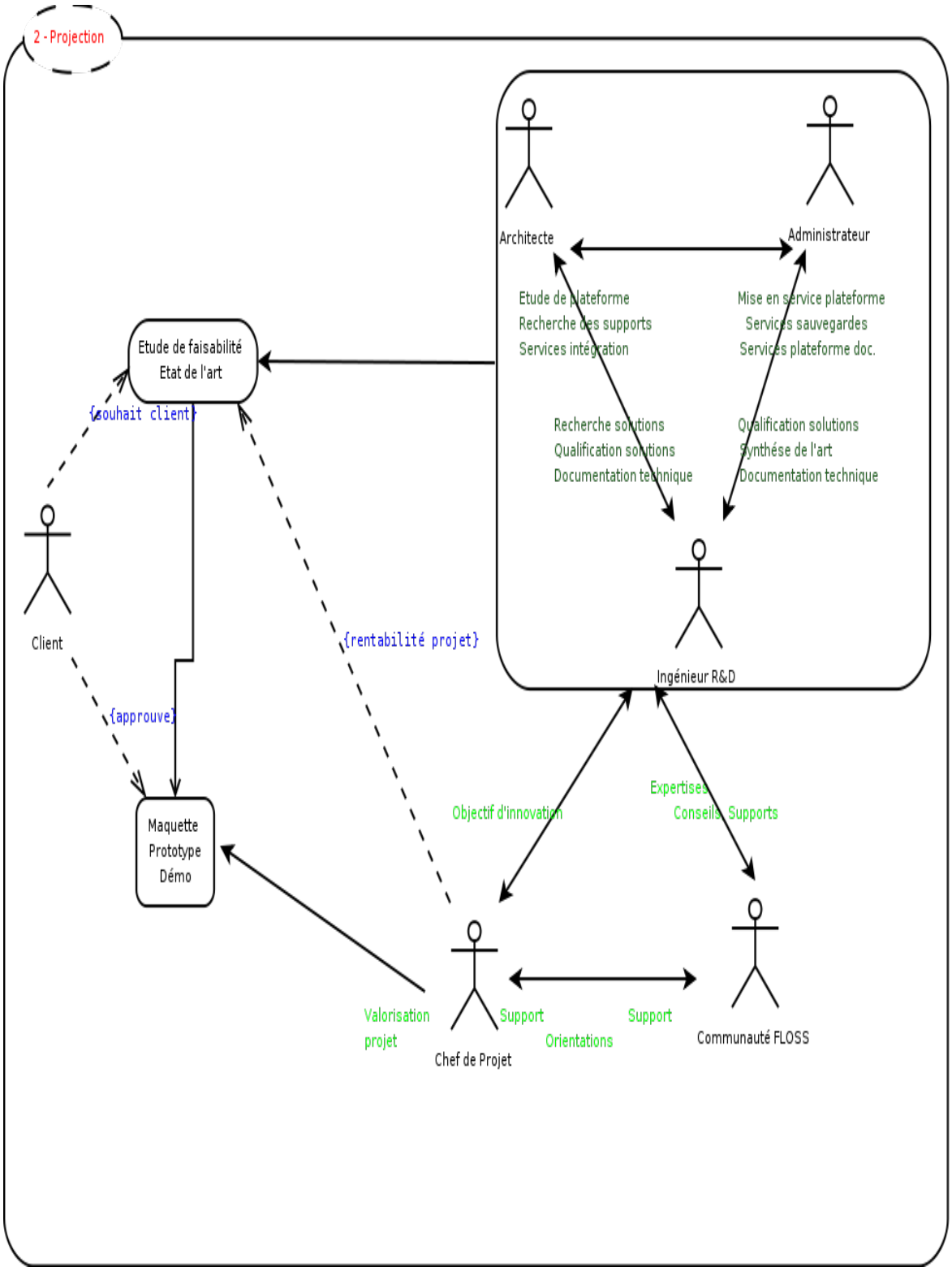
nous conditionnions des réflexes (si SystemT alors utiliser LibZulu + SoftVaporWare) ;

ou ce sont les réflexes qui sont conditionnels (selon que LibZulu accepte les bonnes entrées, donne des bonnes sorties alors l'utiliser dans SystemT)

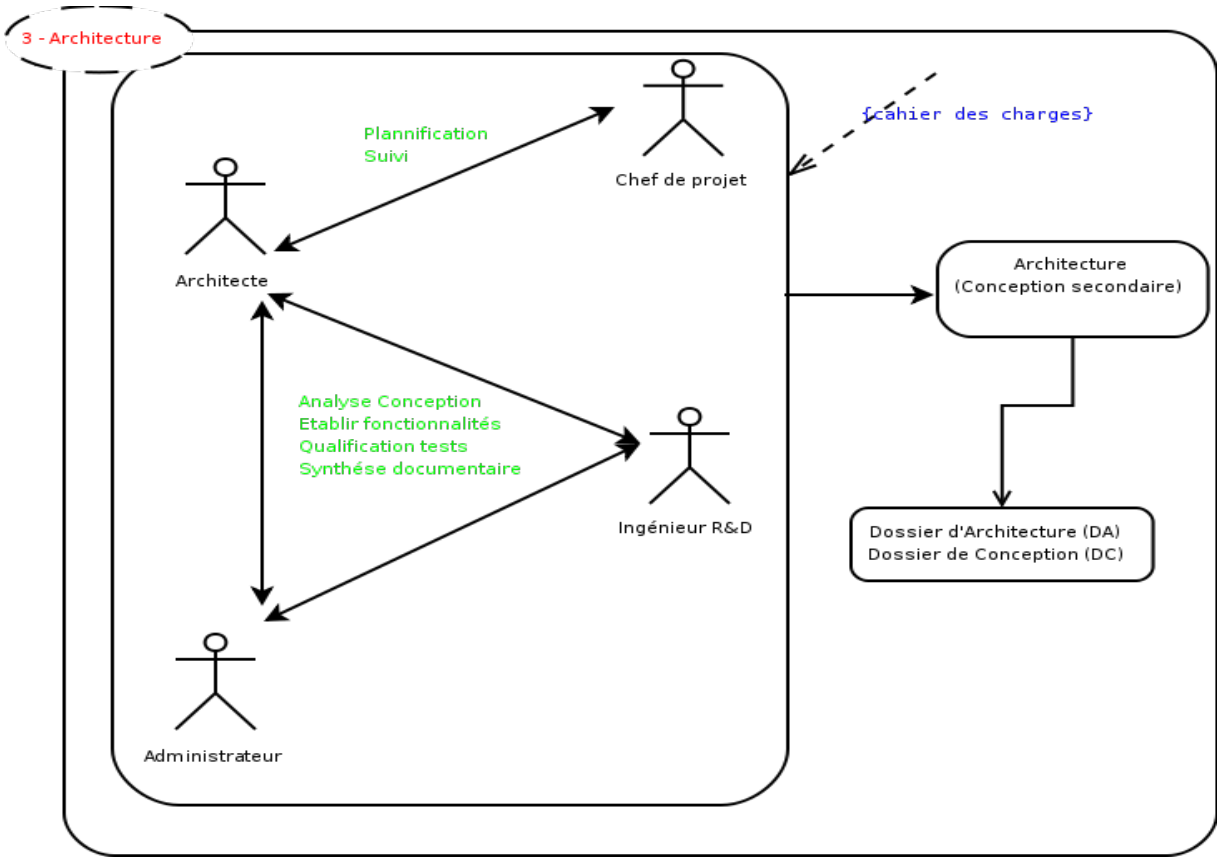
Adapter l'organisation à la R&D FLOSS



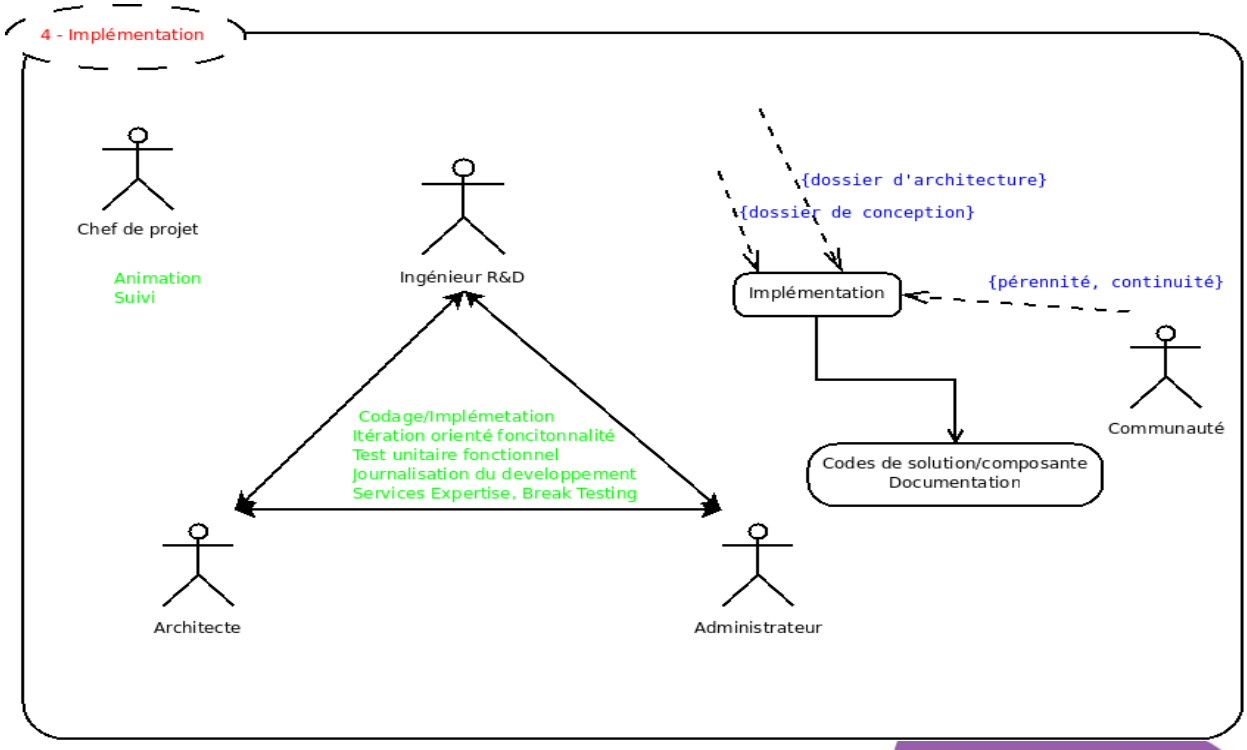
Ecouter-Intervenir-Echanger : Répondre au besoin



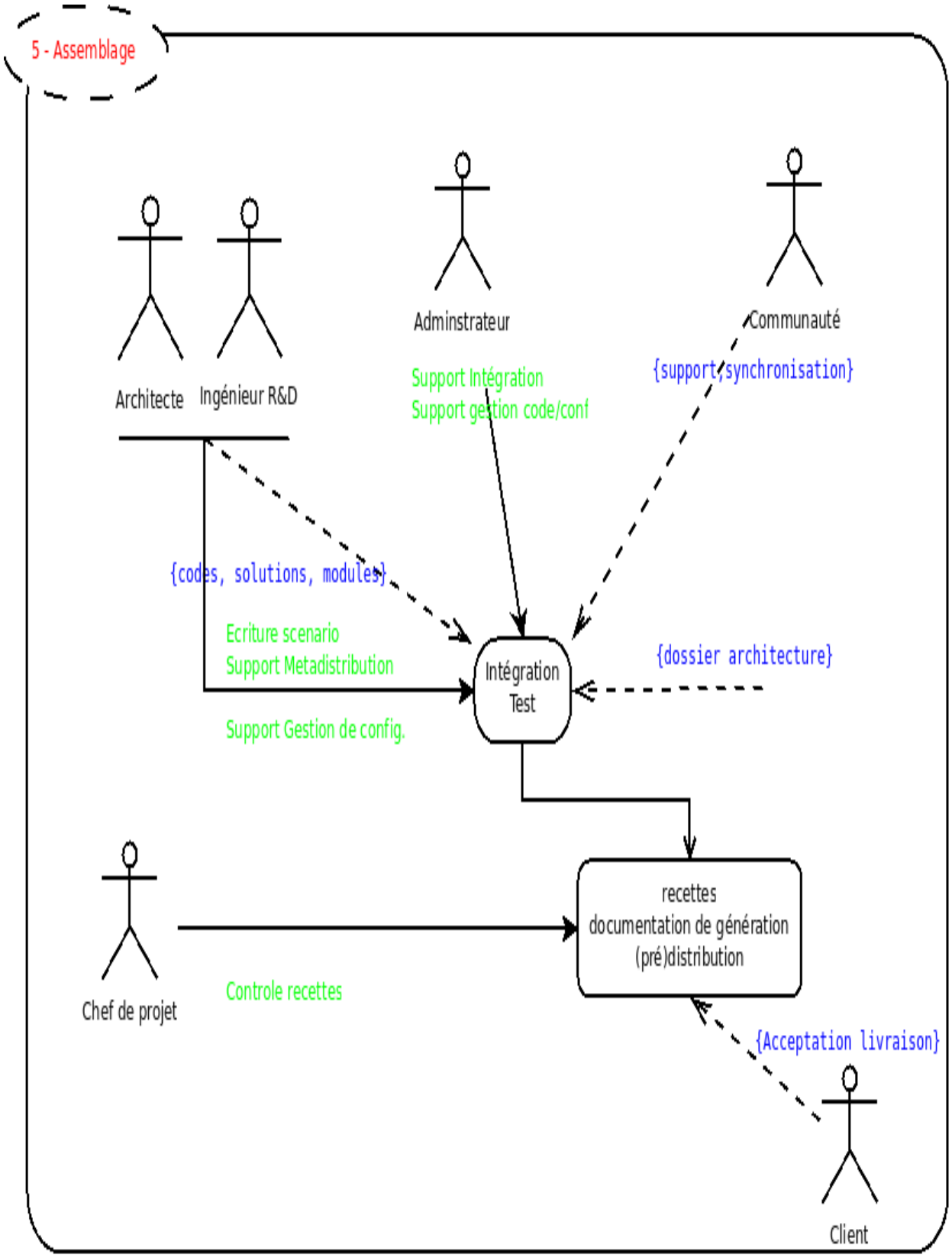
Collecter-Evaluer-Etablir : Estimer une faisabilité

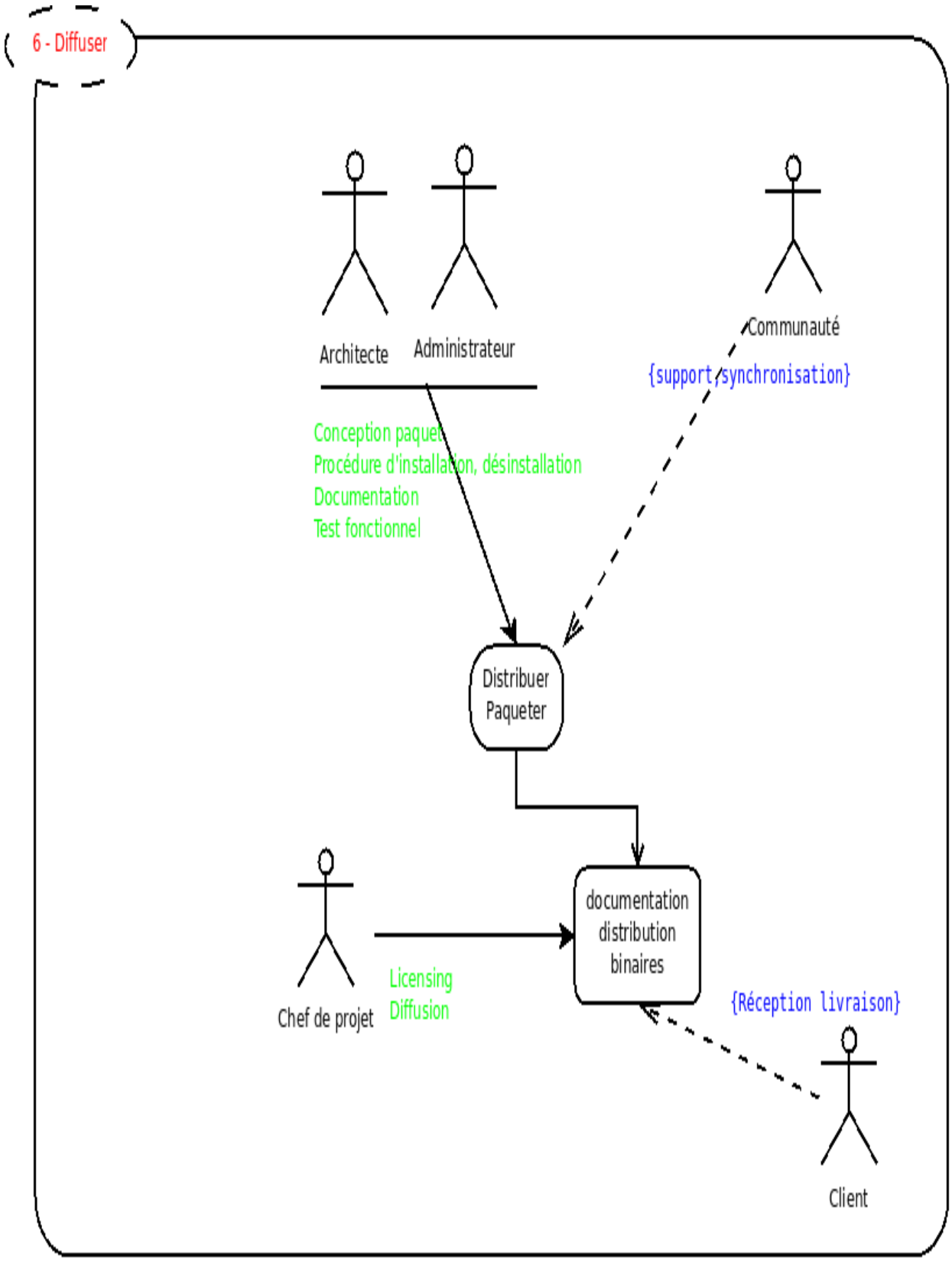


Planifier-Concevoir- Suivi: Baser une architecture



Coder-Tester-Journaliser : Implémenter


5 - Assemblage
Architecte
Ingénieur R&D
Adminstrateur
Communauté
Support Intégration
Support gestion code/conf
{support, synchronisation}
{codes, solutions, modules}
Ecriture scenario
Support Metadistribution
Support Gestion de config.
Intégration
Test
{dossier architecture}
Chef de projet
Controle recettes
recettes
documentation de génération
(pré)distribution
{Acceptation livraison}
Client


6 - Diffuser
Architecte
Administrateur
Communauté
{support, synchronisation}
Conception paquet
Procédure d'installation, désinstallation
Documentation
Test fonctionnel
Distribuer
Paqueter
documentation
distribution
binaires
Chef de projet
Licensing
Diffusion
{Réception livraison}
Client

Elaboration

Associer le maître d'œuvre (si ce n'est pas le rôle du client : associer le client !),

les intervenants de la gestion de projet

aux équipes de développement pour les phases stratégique du projet !

Communication imagée

Simplicité & approche naïve

Découper (sans hacher ou à contrario sans effet tunnel) en sous-projets de quelques mois la R&D logicielle et matérielle

Commentaires

Associer les intervenants

Elaboration :

Le produit d'une R&D finance la R&D suivante (contrainte financière)

Un prototype n'est satisfaisant que s'il est opérationnel (contrainte technique)

Le projet abouti s'il se dispose convenable selon le plan d'architecture (contrainte pilotage)

Pratiquer la métaphore, la structure logique de l'implémentation ne passionne pas tout le monde -> se faire comprendre et non convaincre.

La cohérence d'une élaboration c'est : brut -> raffiné. Un nouveau besoin émerge, temporaire, oneshot ? KISS & naif ! On raffine que si ça Suse ! ;)

Ne pas dépareiller logiciel/matériel comme c'est trop fait.

le matériel a besoin du logiciel pour valider la faisabilité de la carte en prototype .

Le logiciel tourne toujours en définitif sur un matos

Distribution

La diffusion de version prototype même semi validée, jalonne.

Intégrer en continu

mon(Mythe) : le code e(s)t la documentation

ma(Doxa) : la documentation (n')est utile (qu')au client.

(my)Truth : documenting is('nt) evil !

Commentaires

Démystifier : frequent release anyhow

Confirme que le prototype correspond aux spécifications. Les versions, les prototypes forme l'histoire d'un projet, veiller à son cohérence.

Déviances rencontrées :

plus de temps à packager qu'à concevoir la prochaine version.

usure psychologique : prototype kleenex

La release n'est pas testée/évaluées de façon extensive d'une version à l'autre

mauvaise utilisation/perception de l'outil de gestion de révision de code :

-pas faire branche de révision -> mélange du principe sauvegarde du projet, distribution logicielle (stabilisation)

-distinguer sauvegarde de l'environnement de dev. (catastrophe) et le logiciel en R&D

Intégrer en continu ne sert pas tant à se synchroniser dans l'équipe (trop orienté équipe)

qu'à systématiser la construction du SI (mieux ainsi : orienté projet).

croyance : Il y aura pas moins de bug ainsi, il n'en seront pas moins sévère

mais leur détection sera plus rapide. Avec la journalisation d'intégration

Mythe : code ~= doc . code+Epsilon = code à documenter. Doc+Epsilon = code à taper

Ne donner au client que du code ? Non

Ne donner au chef de projet que du code ? Non

Ne donner à l'équipe/équipier que du code ? NON ... sauf.

Par ailleurs écrire une doc :

- cognitivement aide la mémorisation et l'assimilation de connaissance

- techniquement permet de (re-)connaitre les éléments (re-)factorisables

- conceptuellement enracine l'idée que les logiciels sont pour la machines, les fonctionnalités pour l'homme

- aide l'entrée de nouveaux equipiers

Donc le code e(s)t la documentation est une fantaisie, le pb c'est que

ça provoque :

- des docs écrites en fin (pour la distrib) ce qui retardent les bénéfices évoqués ci-dessus.

- des docs retranscrites d'échanges de mails épards non archivés.

Organisation de travail

Maintenir un rythme constant de travail

Veiller à la consistance de ce travail

Présence physique, n-nômes de travail

Commentaires

Organisation de travail

Non le coding sprint ne s'applique pas à l'organisation du projet, mais à l'implémentation de fonctionnalités données du code.

Est il utile d'attendre le prochain "point" pour remonter des inconsistances du planning ?

Réunionite : le mal des dept R&D ? Comme on se pose la question tache1 -> tache2 pour faire le planning,

Se poser la question : est-ce que cette tâche, cette partie du travail nécessite présence permanente en équipe ?

-> Importance de l'autonomie, de la réflexion individuelle, sur soi.

A contrario :

Analyse : l'équipe de projet entière (synergie des métiers, spécialités)

Conception : les architectes (binôme ou trinôme complémentaire. éviter monômes -sauf incompatibilité vicérale-)

Implémentation : en binomes si possible , les architectes , les codeurs habiles, les membres de l'équipe qui s'ennuient, qui veulent se former sur une techno (depuis le logiciel)

Test : en binômes aussi (validation)

Admin/Distribution : monôme possible

Journaliser son activité permet d'être son propre "tracker" et de se responsabiliser (non au role tracker pour les projets)

But n-nômes : vaincre "l'indispensabilité" énoncée ou dans les faits

Pilotage ou Y a-t-il un pilote ?

Tracker ou pas ? Chef de projet alors ?

Euh ... <grosse_bêtise> le client ? </grosse_bêtise>

Réponse(s) ?

un des développeurs. ou un autre.

Commentaires

Pilotage dans le sens maintient du cap sur objectif

Non et non. C'est le meilleur moyen de tuer l'autonomie, la responsabilité collective de l'équipe.

1. Tracker n'est pas un bon pilote (cherche à valider son planning, peu enclin à la remise en question, au refactoring)

2. Chef de projet s'occupe typiquement des "services généraux", le coup de pouce technique est juste envisageable localement & temporairement.

3. Client : noter son numéro de téléphone, car il ne vous rappellera plus.

Initie ou empire le cercle vicieux du :

"Quelqu'un fait cette fonctionnalité ?" et du ;

"Alors ça marche ?"

(Non)Résultat assuré !

L'outillage adéquat, et la perception collective sert de feuille de route.

Simplement un des membres de l'équipe qui est le crédit de l'équipe.

Un développeur d'expérience, de caractère qui UTILISE l'outillage commun

et qui aime à se rappeler pourquoi il fait ce qu'il fait (distanciation)

Pour conclure

1.Transposer les modèles transposables sinon adaptons-les !

2.Elaborer sans clanisme, naïvement. Raffiner utilement et constamment. Aimer trouver certes ! mais chercher

3.Intégrer avec constance. Distribuer sans frénésie. Toujours documenter (par étapes).

4.Equipe à organisation en n-nômes et non par "métier". Responsabilisation & Implication collective, Emulation & Synergie (RIES).

5.**Evoluer au delà du pilotage TopDown**

Outils R&D

Communication

- courriel
- liste de diffusion
- wiki,cms
- forum, groupware

Commentaires

Mail : le tout-mail : livraison interne , début de projet etc => NON ! forwarding à outrance ou au compte goutte : non ! Le mail doit rester un moyen d'alerte (que s'il n'est pas pollué).

Mailing list != mails groupés : Trés mal utilisé , pas toujours a bon escient : pas d'archives disponibles, pas de formatage => ne doit pas suffir pour piloter un projet.

mediawiki : p-e trop lourd pour une équipe réduite , probablement pas aussi adapté que trac,

informations internes à l'entreprise : équipes , listes , moyens (gain de temps pour les nouveaux).

Outils de développement et d'intégration

- éditeur de code , IDE
- outils de debogguage
- émulation, bac à sable
- système de construction
- logiciels de gestion de versions
- metabuild system / meta distribution

Commentaires

Build system : autotools , cmake , scons : tache pouvant devenir très complexe sur des gros projets , exemple gnash : 1 mois)

SCM : avantage de git : merging plus robuste , travail en local , branching facile, admin facile. à contrario de svn qui fini souvent par devenir un logiciel de backup.

Metabuildsystem : Buildroot, OE, Gentoo

Qualité

- système de suivi de bogues
- tests (unitaires et de non-régressions)
- logiciel d'intégration continue

Commentaires

"bug tracking system" : bugzilla, mantis , trac (outil composite)

Tests : cppunit, junit.

"continuous integration" : tinderbox , buildbot (bitten) : risque : se faire piloter par les softs.

Exemple

OpenEmbedded

Buildroot

Pour conclure

La faisabilité d'une R&D FLOSS c'est **prévoir** et **estimer** l'intérêt/pertinence de ces outils.

L'oublier engendre souvent des retards, des "migrations" et une contre productivité croissante au cours du projet.

Ingénierie pour embarqué

BSP

Le BSP, un point crucial du développement embarqué

Définition

Le BSP représente l'ensemble de la solution :

- BIOS
- Bootloader
- Kernel
- RootFS

Commentaires

BSP ? ensemble de la chaîne logiciel embarquée, pas outils/données permettant sa génération.

BIOS+bootload : séquence de boot avant noyau => se cache souvent des verrous cachés comme le format d'image.

kernel+RFS : distribution Linux , ensemble cohérent de logiciel

L'important est de posséder l'ensemble des outils permettant la génération de ce BSP afin de l'adapter.

Importance de la maîtrise de la chaîne de boot

BIOS : OpenFirmware, Coreboot

Coreboot : BIOS libre

Bootloaders avec moniteur : Das U-boot, RedBoot/ecos, Etherboot

Personnalisation :

- rapidité du boot

- adaptation au média
- tests initiaux
- ergonomie

Coûts :

- économie des composants
- économie en licence

Commentaires

Personnalisation :

- rapidité du boot : switch de contexte plus tôt , pas de double initialisation
- adaptation au média : boot + flashage on lan (sagem) , usb -> nand (Nao)
- tests "early" : corruption de flash , récupération en cas de système inopérant
- ergo : user friendliness (splashscreen sur box, affichage LCD, gestion de la télécommande...)

coût :

- économie des composants (eeprom 64ko Coreboot != 512 Ko General Software)

La chaîne compilation

2 chaînes de compilations (croisées, cachées/distantes) :

BIOS/Bootloader et Kernel/Appli.

Glibc ou µClib (et les autres) ?

Librairies auxiliaires...

Hello world ! (dynamically linked, using shared libs, not stripped)

12174 : ELDK

6159 : µClibc

Déboggage ?

Pas de "Stripping"

Flags conseillé :

Commentaires

Génération : eldk, crossdev, buildroot, uboot

296K libuClibc

uClibc code is ANSI/ISO C99 and SUSv3 compliant

autre libs :

512K libstdc++

8.0K libdl

100K libpthread

Debugging pas à pas :

No stripping (section .debug !)

-O0 -fno-schedule-insns -funroll-loops

Pour conclure

Maîtrise, "extensivité", systématique.

Rappelez-vous :

Dimensionner le système de génération de distribution embarqué, c'est encore faire de l'embarqué !

Métadistribution

La distribution

Métadistribution, distribution

La métadistribution fabrique la distribution embarquée.

Utiliser une métadistribution libre permet :

- une indépendance vis à vis d'un constructeur
- une meilleure maîtrise du système
- de capitaliser des compétences et des ressources

Commentaires

Les "métadistributions" des fondeurs fonctionnent mal (PCS Metrowerks, DevRocket Montavista),

les licences empêchent la réutilisation du code,

ce qui empêche le portage de composantes logicielles libre.

OpenEmbedded

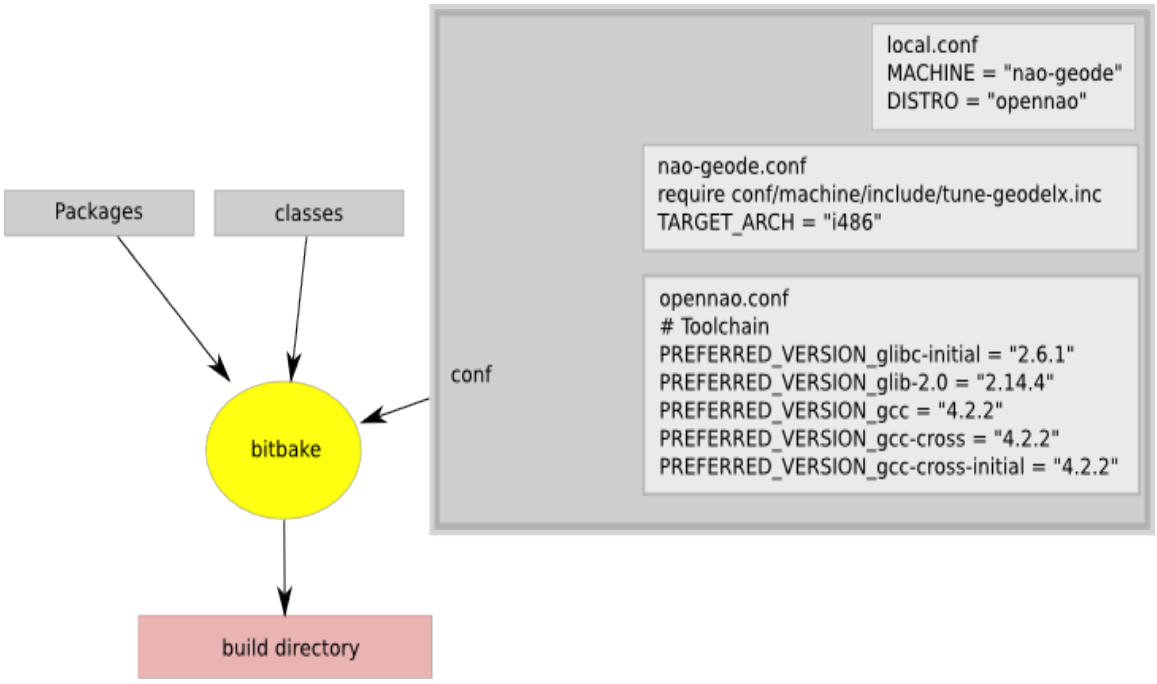


Diagramme simplifié de fonctionnement

Caractère systématique de la génération

Conception pseudo orienté objet : héritage , classe

Tâches adaptées à la création de BSP :

- création de chaînes de compilation croisée
- création d'image du système
- créations de paquets
- gestion d'arbre de paquet

Commentaires

bitbake : task manager , gère les dépendances de taches et l'ordre de leur exécution.

inspiré de emerge (gentoo)

Ecriture des classes en bash & python.

Pour conclure

Pour privilégier le déterminisme et la systématique de la génération :

- fixer le versionnement des paquets,
- centraliser leurs fichiers de configurations,
- conserver les journaux de génération.

Comment dire adieu au longue suites de tests d'intégration !

Embedded Linux

Concevoir son Linux, Fabriquer une image, Débogguer

Comment dimensionner linux ?

- Modulariser
- Démoniser
- Journaliser

Commentaires

Embarquer

Tout module : maîtriser l'empreinte statique mémoire vive.

Dev. module pour tester (inst/desint pour tester)

Spécifier finement le modules.conf et démoniser dans init.d ou udev (ex : gestion energie)

Vérifier avec lsmod et meminfo (support /proc et /sys vivement conseillé)

Utiliser udev/cron pour forcer nettoyage de certains modules (modules plateforme/racines résiduel)

stripper les modules (en prod)

Journaliser et garder des traces (system.map)

Fabriquer son image pour le bootloader

mkimage -A arm -O linux -T kernel -C gzip -a 0x20008000 -e 0x20008000 -n Linux2.6.14 -d zImage uImage-2005-12-15-Thursday

Image Name: Linux2.6.14

Image Type: ARM Linux Kernel Image (gzip compressed) Data

Size: 965280 Bytes = 942.66 kB = 0.92 MB

Load Address: 0x20008000

Entry Point: 0x20008000

Commentaires

Ex Uboot : Pour Connaitre entrypoint (EP), loadaddress (LA/LP) :

Reallocation address and kernel entry point for AT91RM9200ZREALLOC=0x20008000

Déboguer un module

Ne pas faire confiance à un kernel qui OOPS.

1. Augmenter la verbosité (make V=1)
2. Utiliser Ksymsoops (/proc/ksyms vs. system.map)
3. Avoir une sonde JTAG/BDI ?

Commentaires

Modulariser le composant qui plante.

Ksymsoops : Prendre la carte de symbole du noyau et... GDB

Le moniteur de la sonde est indépendant/peu intrusif.

KGDB support & BDI support : mutually exclusive !

Mur de l'activation de la MMU : deboggage avant & après, rien à voir !

Modutils et sécurité

Modules owned by root : refusé ou pas ?

Commentaires

Certaines version embarquée par défaut accepte : NON !

Puis vérifier la phase depmod du kernel : pas de "-r" (accepte les modules non root)

Dimensionner linux

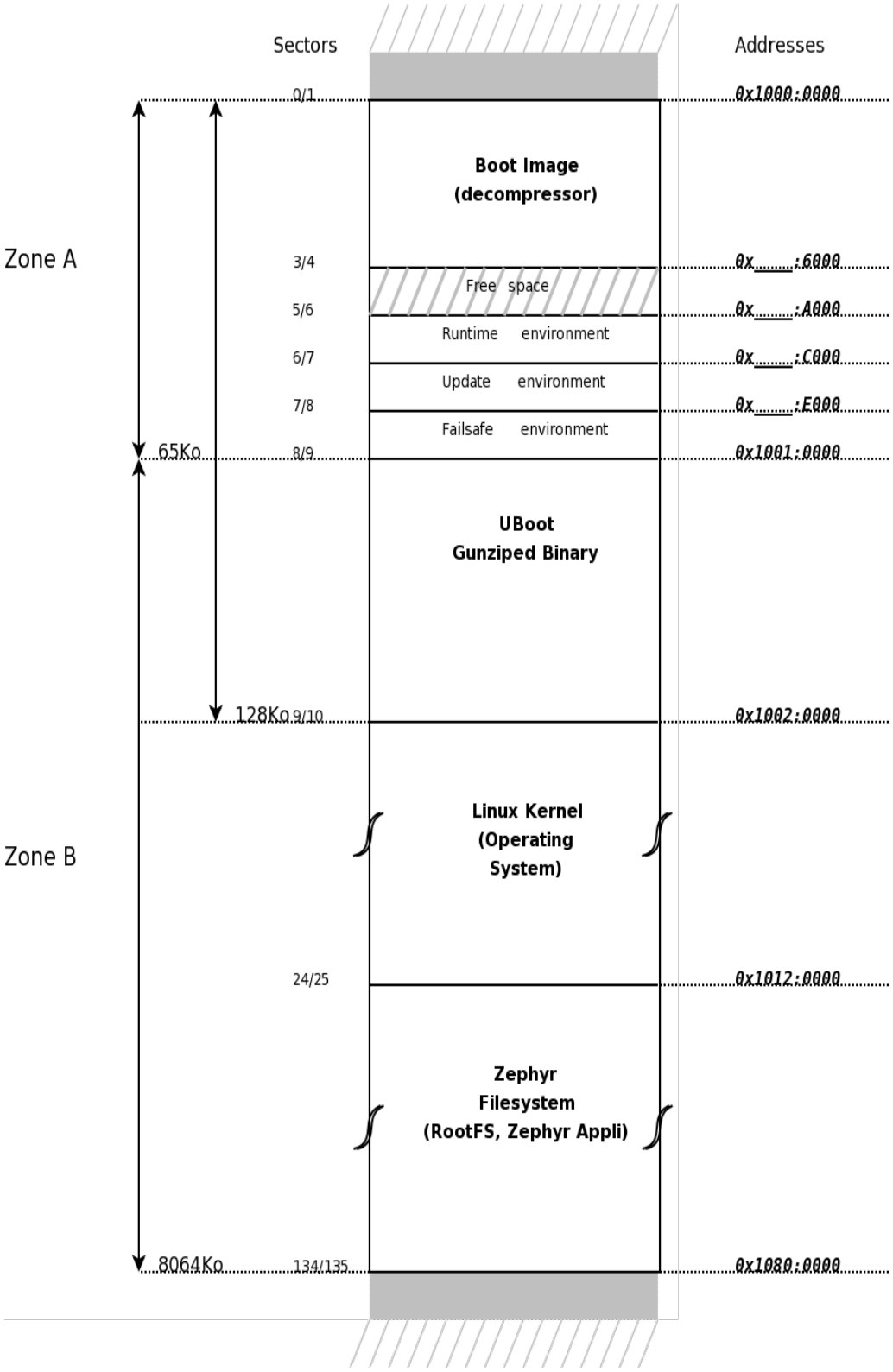


Diagramme mémoire Flash

Commentaires

Orga flash

Importance du partitionnement flash

Conserver de l'espace pour le boot loader et son environnement.

Duplication d'environnement (ex : failsafe)

Pour conclure

Passer un peu (plus, beaucoup, ... ;) de temps pour embarquement le kernel,

pour hacker les interfaces, y ajouter des supports matériels, enlever des branches "mortes" :

Dédier.

Questions ? Animaux poilus ?

Contact

Deschamps Mathieu (mathdesc)
mathdesc@scourge.fr
http://www.scourge.biz