Leonardo de Larrosa

Développeur principal Javascript / Typescript

Front-End / Full-Stack

Digital Profile https://leonardolarsan.github.io/CV

Contenu

Profil

Les rôles

- Développeur Javascript / Typescript Full-Stack / Front-End
- Développeur React
- Développeur Node.js
- Développeur Vue
- Développeur natif React
- Développeur Ionic Vue

Aptitudes

- Expérience exceptionnelle dans l'analyse, la conception et le développement d'applications Web.
- Recherche constante de nouvelles technologies.

- Grande capacité à anticiper les problèmes et à les résoudre.
- Codage lisible, évolutif, réutilisable et efficace.
- Forces pour organiser et estimer les temps de travail.
- Orienté vers les résultats et la prise de décision en temps opportun.
- Compétences pour interagir avec les clients et les utilisateurs.
- Facilité à transmettre les connaissances et à favoriser le travail en équipe.
- Mise en œuvre des pratiques modernes et des dernières tendances de développement Web.
- Idées et opinions visant à éviter la dette technique à long terme.
- Création de tests unitaires et d'intégration.
- Développement d'outils d'automatisation pour les tâches quotidiennes.

Connaissances et compétences

Langages de programmation

- Javascript
- TypesScript
- HTML5-CSS3 Responsive

React

-React.js -react-create-app -React Native -Redux -Mobx -Class Components -Hooks Components -Higher-Order Components -Styled Components -SASS -Next -Axios -Mocha -Chai -Jest -Enzyme -React Testing Library -Material UI -Native Base

Vue

- Vue.js
- vue-cli
- Vuex
- Vue.observable()
- Styled Components
- Tailwind CSS
- Post CSS
- SASS
- Directives
- Vue PWA
- Capacitor
- Nuxt
- Native Script Vue
- Ionic Vue
- Vuetify
- BootstrapVue
- Element
- Framework7 Vue
- Axios
- Vue Test Library

Node

- Express.js
- express-generator
- Nest
- Micro.js
- Adonis
- Mongoose
- Socket.io
- Open Api
- JWT
- Mocha
- Chai

Base de Datos

- MongoDB
- IndexedDB
- Redis
- CouchDB
- PouchDB
- MySQL
- MariaDB
- SQLServer

Test E2E

- Cypress

Philosophies et pratiques de développement

- Clean Code
- TDD
- SOLID
- KISS
- DRY
- GIT Flow
- LEAN

Outils de développement

- Visual Studio Code
- GIT
- GIT Kraken
- SQLectron
- MySQL Workbench
- Data Grip
- Robomongo
- Robo 3T
- Postman

Architectures de programmation et modèles de conception

- MVC
- MVP
- MVVM
- Factory
- Flux
- Reducer
- Clase
- Observer
- DAO
- Services
- Repoitory
- Container
- Component

Méthodologies agiles, outils d'analyse et organisation

- Star UML
- Kanban
- Scrum
- Scrumban
- Slack
- Trello
- Jira

Design graphique

- Gimp
- Inkscape
- Krita

Systèmes d'exploitation

- Linux(Manjaro)
- Windows

Bureautique

- Libre Office
- Microsoft Office

Langues

- Anglais technique
- Espagnol maternel

Données

Nom complet:

Leonardo Gabriel Larrosa Sánchez

Date de naissance:

15-10-1988

Nationalité:

Argentin

État civil:

Celibataire

Adresse:

Suarez 1 ■■■ 4e 11, La Boca, Ciudad de Buenos Aires

CUIL:

■■ - ■■■■■■■■ - ■

Téléphone:

11-2338-0144

E-mail:

leogab.larsan@gmail.com

Linkedin :

https://www.linkedin.com/in/leonardolarrosa/

Éducation

Udemy

Vue.js

25/12/2017 - 14/01/2018

- Vue.js
- Vue Instance
- data
- methods
- computed and filter
- Directives
- Transitions
- Components
- Props
- component lifecycle
- vue-router
- vue-cli
- Vue2

Udemy

Réagir.js

11/05/2017 - 12/20/2017

- React.js
- Props
- State and Lifecycle
- JSX
- Stateful and Stateless
- Components
- React Router
- Wrap Components
- Layout Components
- HOC Components
- Redux
- CSS
- Animations width dinamic classNames

- Axios
- webpack
- create-react-app
- React Native

UDACITÉ

Analyse et gestion de projet

02/2016 - 04/2016

- Kanban
- Scrum
- XP
- Entrevistas
- UML
- Microsoft Project

Centre de formation professionnelle n°27 Luz y Fuerza

Pile de développeur MOYENNE

04/2015 - 11/2015

- Javascript
- jQuery
- AJAX
- JSON
- Mustache.js
- Angular.js
- mongoDB
- Express.js

Centre de formation professionnelle n°27 Luz y Fuerza

Développeur Java

04/2015 - 11/2015

- NetBean
- Développement Java pour ordinateur de bureau
- Développement JAVA avec Database en utilisant le pattern DAO
- Service Web Java (JSP)

Centre de formation professionnelle n°8 SMATA

Concepteur et développeur Web Full-Stack

04/2014 - 11/2014

- HTML5CSS3: responsive and animation
- GIMPHTML5-CANVASHTML5-SVG
- XAMPP
- .htacces (URL amigables)
- PHP (Estructurado y Modular)
- PHP POO (orientado a objetos)
- PHP-POO-MySQL
- Javascript
- jQuery
- Ajax
- JSON
- WebSocket

Centre de formation professionnelle n°8 SMATA

Architecture, construction et administration de bases de données relationnelles

04/2012 - 06/2013

- Théorie des bases de données relationnelles
- MySQL

- OracleDB
- PL/SQL
- Architecture et construction de bases de données
- Administration des bases de données

Centre de formation professionnelle n°8 SMATA

Analyste - Programmeur Java

04/2012 - 11/2012

- Introduction à la programmation avec Python
- Programmation structurée et modulaire avec Python
- Théorie des objets : classes, attributs, méthodes et héritages avec Java
- Langage de modélisation unifié UML avec StarUML
- Organisation de projet avec ProjectLibre

Centre de formation professionnelle n°8 SMATA

Opérateur d'images numériques

04/2012 - 11/2012

- Adobe Photoshop
- Adobe Illustrator
- CorelDraw
- Photo-peinture Corel

Projet Programar.org

Développeur web

04/2011 - 11/2011

- Notepad++
- HTML4
- CSS3
- XAMPP
- PHP
- .htaccess
- Javascript
- Ajax
- MySQL
- FileZilla
- Ubuntu
- LAMPP
- Anglais technique
- Scrum
- Organisation du projet

École de commerce Nº4

Expert Mercartil

04/2001 - 11/2006

- Comptabilité
- Administration d'entreprises
- Microéconomie
- Macroéconomie

Expérience de travail

Siembro (10/2020-actuel)

Chef d'équipe front-end

Leadership et développement des nouvelles applications Back-Office et Front-Office en utilisant les bonnes pratiques, des architectures propres et des modèles de conception modernes.

- React.js
- Ionic React
- Styled Components
- React Hooks
- React Test Library
- Axios
- Typescript

Digital House (6 / 2019-10 / 2020)

Front-End React / Vue Javascript / Typescript

Développement et maintenance de Playground, une Web App pour Digital House, utilisant les technologies suivantes: - React.js - React Hooks - Styled Components - Redux - Ajax - SASS - Monaco, - Jest y Enzyme

- Maintenance et développement du site Internet de la MDigital House:
- Nuxt.js
- Vue
- Vuex
- SASS
- Vue Test Utils

Cablevisión Flow (11 / 2018-6 / 2019)

Front-End React Javascript

Développement de Flow Web Client, la célèbre Web App pour la télévision numérique, et Radio flow, une App pour Smart TV qui permet de syntoniser et d'écouter des radios, en utilisant les technologies suivantes: - React.js - Javascript - Mobx - React Hooks - SASS - Mocha - Chai - Jest - Enzyme y React Test Library

RCN (10 / 2016-11 / 2018)

Full-Stack Javascript/Typescript

Analyse, conception et développement d'applications web et mobiles à usage interne de l'entreprise. Les outils utilisés étaient: - Javascript - Node.js - Express.js - PM2 - GIT - React.js - React Native - Typescript - Vue.js - Material UI - Vue Material - Native Base - Bootstrap - MongoDB - SQLServer - IndexedDB - WebSocket - Ajax - Fetch y Axios

FreeLanceur (11 / 2011-09 / 2016)

PHP-MYSQL-Angular.js à pile complète

Analyse, conception et développement d'applications Web et mobiles au sein d'une équipe freelance de 5 personnes, que j'ai fondée, dirigée et participée pendant plus de 4 ans en utilisant les technologies suivantes : - PHP5-MySQL - HTML-CSS - Javascript Vanilla - JQuery - Phonegap - NW.js(Node Web Kit) - Angular.js

Sur moi

Brève description (de moi-même)

Argentin, 32 ans. Je vis au jour le jour en essayant d'être bon dans mon travail et avec les autres. Je vis avec mon partenaire depuis plus de 7 ans, et avec mon chat siamois qui a plus de 10 ans. Pendant mon temps libre, quand je ne travaille pas sur mon ordinateur, je suis toujours sur l'ordinateur, mais je joue à des jeux en ligne, je conçois des plans de maison en 3D ou je regarde des séries ou des documentaires sur la science, l'architecture, la médecine, etc. J'aspire à avoir ma propre maison et à fonder une famille, un rêve qui est très difficile à réaliser. Je vis en location en continu, ne pouvant pas construire une histoire en un seul endroit, car je suis locataire et déménager est quelque chose de normal dans ma vie. En bon millennial, je comprends ce que signifie la « modernité liquide » dans laquelle nous vivons, où tout est incertain et éphémère. J'accepte les changements. J'ai mis de côté une vie monotone mais instable à la longue. Je sacrifie mon présent en travaillant dur, en échange de la stabilité que je peux obtenir dans le futur.

Description professionnelle

Développeur full stack Javascript et Typescriopt et responsable technique, avec plus de 10 ans d'expérience. Ma pile principale est Node.js avec Typescript, React, SASS, Ionic, Vue, Express, Fastify, Express, mongoDB, Redis, MySQL, Jest et Cypress, PM2 et Docker. J'ai plus d'expérience dans le Front-end. J'insiste sur la qualité du travail, sa planification, son organisation, sa résolution, sa prise de décision et son leadership.

Réflexions et enseignements qui ont marqué mon évolution en tant que développeur

Pourquoi miser sur un code de qualité et non sur une productivité immédiate ?

On voit souvent que les développeurs sont contraints d'augmenter leur productivité avec des délais très serrés, ce qui les amène à tomber dans de mauvaises pratiques de développement qui ruinent la qualité du code. Face à cette situation, les solutions rapides, et pour l'instant, sont faciles à mettre en œuvre au début des projets, mais lorsqu'elles prennent de l'ampleur, et qu'un plus grand nombre de fonctionnalités sont ajoutées, elles augmentent en complexité, au point de devenir trop difficile pour un développeur de maintenir ces projets lorsque le code écrit est illisible, encombré et imprévisible.

Ce problème est connu sous le nom de « dette technique » tant redoutée et, bien qu'il s'agisse d'un sujet bien connu dans le développement de logiciels, de nombreux responsables techniques et directeurs techniques sous-estiment grandement ce problème. Ils pensent que des solutions ou des développements rapides, axés uniquement sur la fonctionnalité et non sur la qualité, sont synonymes de productivité élevée, car ils peuvent livrer rapidement de nouvelles fonctionnalités aux clients. Cependant, lorsque le code de mauvaise qualité augmente en taille, il devient exponentiellement plus difficile à maintenir, et cela se traduit par une perte de productivité dans le secteur du développement, et ils essaient de résoudre ce problème en mettant encore plus de pression sur les développeurs pour atteindre les niveaux de productivité. qu'ils avaient au début des projets, quand ils étaient encore maintenables, de sorte que les développeurs finissent par être stressés, démoralisés et épuisés mentalement. Ce rythme de travail ne peut être soutenu par les développeurs que pendant une période limitée, après quoi leur productivité chute considérablement car ils se retrouvent atteints du syndrome d'épuisement professionnel.

Essayer d'embaucher de nouveaux développeurs pour résoudre cette situation est également une erreur. Les nouveaux membres de l'équipe, désireux de contribuer, devront faire face au désordre et au chaos des projets. Et, étant donné la nécessité d'être décisif, et puisqu'il n'y a pas de modèles, d'architecture ou quoi que ce soit de défini, ils essaieront de résoudre leurs problèmes comme bon leur semble et, par conséquent, ils introduiront des solutions rapides qui augmenteront encore la dette technique existante. . Le temps continuera à avancer, et avec lui la complexité des projets et leur dette technique. La productivité baissera tellement que, même s'ils intègrent de nouveaux membres de l'équipe, il y a peu de choses qui peuvent continuer à progresser, car, avec le moindre changement ou ajout dans le code, il faudra investir beaucoup d'efforts pour résoudre le nouveau erreurs introduites. Dans ce type de cas, comme malheureusement la dette technique n'a jamais été « payée », les projets entrent en faillite technique, et c'est alors que les secteurs informatiques décident de migrer les projets vers les nouvelles technologies, accusant les technologies précédentes d'être la cause des échecs. . . .

Tout ce qui précède arrive très souvent parce que les dirigeants ont tendance à sous-estimer la valeur du travail technique et sa qualité, aussi parce qu'ils ignorent ou ne comprennent pas comment les bonnes pratiques améliorent la qualité du code et, face à ce manque, naturalisent l'exponentielle difficulté qui survient au jour le jour, au cours du processus de développement du logiciel.

Malheureusement, de nombreux dirigeants justifient leurs mauvaises décisions en arguant que pour écrire du code de haut niveau, il serait nécessaire d'investir plus de temps et d'efforts pour réaliser de nouvelles fonctionnalités, et que cela entraîne des pertes de productivité dans le développement. Cependant, même s'ils ont raison à court terme, puisque les nouvelles fonctionnalités finissent par être livrées plus tard, ils ignorent totalement qu'à moyen et long terme, un code ingérable devient exponentiellement plus difficile à maintenir. Par conséquent, un code de haute qualité s'avère plus facile à maintenir et plus productif dans la plupart des projets d'aujourd'hui, car ceux-ci augmentent en taille et en complexité au fil du temps.

Pourquoi vaut-il parfois mieux privilégier la qualité plutôt que les méthodologies ?

Habituellement, lorsque les projets démarrent et sont maintenables, les méthodologies agiles et les processus de développement formels ne sont pas pris en compte. Celles-ci ont tendance à devenir importantes tardivement, lorsque le code est déjà devenu non maintenable en raison de son augmentation de la taille et de la complexité, et génère une faible productivité.

Je pense que c'est une erreur, dans ce type de situation, que les méthodologies soient privilégiées par rapport à la qualité du code. Au début du projet, la méthodologie, l'organisation et les processus de développement aident mais, au fil du temps, la dette technique progresse, et bien que d'énormes efforts soient faits pour éviter les erreurs avec le secteur QA, et que les problèmes d'estimation soient tentés de résoudre. en affinant l'organisation, la planification et les processus de développement formels, à chaque sprint, il y aura de plus en plus de retards et de bogues. Tous les efforts qui sont faits en dehors du code pour compenser les lacunes du code ne fonctionnent pas, car c'est comme les ordures qui se cachent sous le tapis, les ordures qu'est le code vont continuer à s'accumuler et le tapis à un moment donné ne suffira plus à couvrir autant de déchets. La meilleure chose à faire est de réparer tout ce gâchis et de ne pas essayer de le dissimuler.

Les problèmes de code sont résolus en travaillant sur le même code, en améliorant sa qualité par étapes, petit à petit avec des refactorisations successives, en appliquant les bonnes pratiques jusqu'à obtenir un code propre et maintenable.

Pourquoi est-ce que je préconise le Clean Code et pas d'autres pratiques de développement ?

Clean Code est une philosophie de développement logiciel, avec une liste de règles et un ensemble de bonnes pratiques de développement, dont beaucoup proviennent d'ailleurs, qui facilitent l'écriture et la lecture d'un code, le rendent plus facile à comprendre et, par conséquent, assurent sa maintenabilité. Contrairement à d'autres pratiques plus difficiles à mettre en œuvre, comme le TDD qui nécessite une grande connaissance des tests, Clean Code peut être appliqué par tous les développeurs, rien qu'en appliquant quelques règles simples la viabilité d'un projet à long terme peut être assurée. Un code propre et élégant est facile et peu coûteux à faire évoluer, à faire évoluer et à maintenir.

Pourquoi est-ce que je préfère implémenter du Clean Code et pas une bonne documentation ?

La documentation ment souvent ou ne dit pas toute la vérité. Il s'agit d'un texte qui, fréquemment, est obsolète par rapport à la réalité ou ne reflète pas la fonctionnalité du code qui a subi des modifications répétées. Il peut également arriver que les développeurs ne tiennent pas à jour la documentation. En fin de compte, il n'y a pas de meilleure source de vérité que le code lui-même, car le code s'exécute et fonctionne ou ne fonctionne pas. Pour cette raison, il est important que le code soit lisible, prévisible et maintenable. Toute règle métier complexe qui a été oubliée peut facilement être mémorisée en lisant un code bien écrit.

Pourquoi un langage typé et pas dynamique ?

Il est typique de voir un développeur JavaScript déboguer, devoir exécuter son application et insérer "console.log" dans le code pour voir dans la console si les données sont envoyées correctement, ou si elles arrivent dans le bon format, etc. . Les langages dynamiques lorsqu'ils sont utilisés correctement s'avèrent très pratiques, mais lorsqu'on abuse de leurs vertus et de leur flexibilité, ils peuvent générer du code imprévisible, illisible et très sujet aux erreurs.

Malheureusement, les programmeurs qui n'ont utilisé que des langages dynamiques ne savent pas comment et quand il est correct d'utiliser les vertus de ces langages, et lorsque cela se produit, ils créent un code dynamique très difficile à maintenir. Les langages typés viennent résoudre ce problème, car ce sont des langages bien documentés et prévisibles, et ils sont très stricts dans la façon dont ils traitent les données.

Les partisans des langages dynamiques diront que les langages typés ajoutent une couche d'abstraction inutile à la gestion des données, et que cela nuit à la productivité. C'est assez vrai car au début il faut investir du temps pour définir comment vont être les données, mais à moyen et long terme, ces langages permettront d'éviter la plupart des erreurs, sans avoir besoin d'exécuter le code en direct. ce qui est également très favorable lorsqu'il s'agit de faire des refactors.

Valeurs par défaut selon leur type et non des résultats dynamiques.

Plusieurs fois, j'ai dû consommer des API créées par d'autres développeurs, qui ont malheureusement répondu avec des données difficiles à traiter, soit parce que les objets étaient dynamiques, soit parce que les propriétés des objets variaient dans leur type de données, soit parce que ces objets variaient dans la quantité de propriétés qu'ils contenaient, etc. Cela m'a obligé à développer en permanence de manière défensive, c'est-à-dire que je devais demander dans le code, en utilisant des conditions, si les données existaient et si elles avaient le bon format pour éviter les erreurs d'exécution, et si elles n'étaient pas dans le bon format, Je devais le reconvertir ou l'obtenir de l'autre côté de la réponse, une question qui a fini par gâcher mon code avec l'utilisation de nombreuses conditions qui ont empêché les erreurs et qui, d'une manière ou d'une autre, ont essayé d'obtenir les données à capable de le gérer sur la base de décisions, c'est-à-dire plus de conditionnels.

Un autre problème qui m'a ajouté de la complexité technique s'est produit lorsque les données n'avaient pas un nom très descriptif de ce qu'elles signifiaient, encore

moins pouvais-je deviner de quoi il s'agissait, ou pire encore, lorsque les données ne déclaraient pas de conclusions ou de déductions, et Moi-même avec des conditionnels, je devais savoir de quel type de données il s'agissait.

Je considère que placer des valeurs par défaut, c'est respecter le type de données d'une variable ou d'une propriété, afin qu'elle conserve toujours le même type de données, même lorsqu'une valeur vide a été tentée pour lui attribuer, elle doit continuer à conserver le même format . Quand je parle de valeurs par défaut vides selon leur type, je veux dire par exemple qu'une propriété qui est une liste d'objets ne doit jamais avoir une valeur autre qu'une liste, et si la liste est vide, elle ne doit pas être null , à la place, il devrait s'agir d'une liste sans valeurs.

La même chose se produit avec les propriétés qui sont des nombres, qui ne devraient pas cesser d'être des nombres à tout moment, et en cas de vouloir représenter une valeur vide par défaut, ce sera 0. Et quant aux propriétés qui sont du texte, elles doivent avoir une valeur vide text et non un null, loin de là, un autre type de données.

En raison de la mauvaise pratique des autres, la complexité technique que cela ajoutait à mon code était telle que ma productivité était considérablement réduite. Heureusement, j'ai appris, lors d'une rencontre, une stratégie pour le résoudre, en normalisant les données dans la couche où elles entrent dans l'application. Je n'ai plus à essayer de corriger les données, ou d'éviter les erreurs dans d'autres parties de l'application, j'ai appris à résoudre la plupart des problèmes liés à la mauvaise gestion des données.

Pourquoi Clean Architecture et pas n'importe quelle improvisation ou solution miracle ?

Il est très fréquent de voir dans des projets, dans lesquels une maintenance est nécessaire, que les différents types de fonctionnalités de code ne sont pas regroupés, classés, ou ordonnés, et même qu'ils sont mélangés avec d'autres types de fonctionnalités. Cela rend difficile de savoir où placer les nouvelles fonctionnalités et, de plus, qu'il n'est pas très intuitif où trouver les fonctionnalités déjà réalisées.

Une autre situation fréquente est de voir que les projets n'ont pas un modèle unique pour exécuter une certaine fonctionnalité, ou que les différentes parties de l'application communiquent les unes avec les autres sans hiérarchie claire, ni n'ont-ils un flux intuitif dans la façon dont ces parties communiquent. . Tous ces types de situations réduisent la productivité de tout un secteur du développement.

Clean Architecture vient résoudre tous ces problèmes liés à l'organisation du code. Il hérite de toutes les pratiques du Clean Code, centrées sur la qualité du code, mais les porte à une échelle supérieure pour les projets, en ordonnant, classant et définissant les flux de communication de chacune de ses parties, pour parvenir à une homogénéisation correcte. Il établit des règles claires qui s'avèrent très intuitives pour les développeurs lorsqu'ils doivent ajouter et modifier des fonctionnalités au projet.

Il est vrai qu'avec un MVC simple et bien dimensionné, MVP, MVVM est largement suffisant, mais il existe d'autres architectures qui, bien qu'elles soient plus complexes à comprendre et à apprendre, sont idéales pour des projets qui sont vraiment énormes et qui doivent durer un long moment.

Pourquoi GitHub Flow et pas GitLab Flow ?

L'utilisation de GitLab Flow a ses avantages et ses inconvénients. Il permet de s'attaquer à toutes les erreurs de code avec l'utilisation de plusieurs branches de développement. Cependant, il est également très probable qu'il y ait des différences et des conflits de code entre les branches. Cela implique que les développeurs doivent continuellement investir des efforts pour résoudre les conflits, au risque de les corriger de manière incorrecte. Cette difficulté est encore plus accentuée lorsque la méthodologie agile utilisée est SCRUM, car les livraisons se font après un temps de développement long et les différences de code et de conflits sont plus importantes.

GitHub Flow, en revanche, nous offre un workflow beaucoup plus simplifié, avec une seule branche de développement comme la vraie, et chaque changement ou modification qui y est apporté doit être traité comme s'il s'agissait du produit final. Au lieu de consacrer autant d'efforts à résoudre les conflits de code, vous les consacrez à l'amélioration de la qualité de votre nouveau code.

GitHub Flow encourage les développeurs à créer de petites versions continues de haute qualité, ce qui est idéal lorsque vous souhaitez intégrer une « intégration continue » dans votre environnement de production, sans avoir à attendre longtemps que le client voit les nouvelles avancées de votre produit.

Suivant la même ligne de Kaizen, GitHub Flow propose de conquérir de petits objectifs pour atteindre un objectif final plus grand, il s'intègre donc naturellement à Kanban et Scrumban, des méthodologies également extrêmement agiles et basées sur de petites tâches livrables.

Pourquoi Kanban et pas SCRUM ?

Scrum est sans aucun doute une excellente méthodologie lorsqu'elle est appliquée correctement, mais elle est vraiment difficile à mettre en œuvre pour la plupart des entreprises, en particulier celles qui sont petites et moyennes.

La difficulté de la mise en œuvre de SCRUM est qu'il n'est pas préparé à faire face à des changements brusques au cours d'un Sprint. L'analyse, la planification, l'organisation et l'estimation des tâches sont effectuées avant le début du Sprint. L'ajout d'une tâche qui n'était pas prévue amène le chaos. Lorsque les développeurs sont en période creuse pour des raisons imprévues, ils ont tendance à générer des solutions rapides et de mauvaise qualité, accumulant ainsi une dette technique qui affectera la productivité à l'avenir.

SCRUM ne doit être utilisé que dans les grandes équipes de travail, car de nombreux rôles supplémentaires et spécifiques sont nécessaires pour pouvoir mener à bien la méthodologie agile.

Kanban partage des similitudes avec SCRUM, mais son approche est opposée, car il n'a pas de sprints rigides et hautement planifiés. Kanban reçoit les tâches et leur planification fait partie du statut de la tâche elle-même, c'est-à-dire que ceux qui créent, planifient, priorisent et conçoivent les tâches participent également au tableau avec le statut de celles-ci. Par conséquent, de cette manière, il est facile de voir comment les tâches se déroulent entre les différents états, et il peut être détecté dans quels états il y a des goulots d'étranglement ou d'autres problèmes.

Grâce au fait que Kanban a des états pour décrire dans quelle phase se trouvent les tâches, et qu'il ne nécessite pas de réunions longues et approfondies (comme la planification SCRUM), les réunions de planification se font par tâche séparément des autres, c'est-à-dire qu'elles sont des réunions beaucoup plus courtes qui n'occupent pas des journées entières de travail.

Kanban est aligné sur la pratique Kaizen qui consiste à diviser les tâches complexes en objectifs petits et incrémentaux, où chaque tâche qui représente un objectif ne peut pas durer plus de deux ou trois heures de travail, et si c'était le cas, l'objectif doit encore être divisé davantage sur tâches plus petites et plus concrètes. Grâce à cela, il est possible d'estimer combien de temps de développement une user story peut prendre, en comptant le nombre d'objectifs et en les multipliant par deux ou trois heures.

En ce qui concerne les changements inattendus et de dernière minute qui surviennent inévitablement, avec Kanban, ils peuvent être intégrés de manière transparente. Chaque changement sera considéré comme une nouvelle tâche. Ensuite, il convient de déterminer quelles tâches doivent être résolues de manière plus urgente que d'autres et seront prioritaires.

Kanban est également approprié pour les grandes équipes de travail car il s'adapte très bien en leur sein. Il permet de visualiser toutes les tâches de tous les secteurs, ou d'un en particulier, en les filtrant par un système d'étiquetage, et chaque secteur ou groupe de travail peut avoir ses propres étiquettes pour identifier plus facilement ses tâches.

Comme si cela ne suffisait pas, les règles Kanban étant très concrètes et simples, des pratiques issues d'autres méthodologies peuvent être intégrées. Il est très courant que certaines pratiques SCRUM soient incorporées en raison des besoins des entreprises et des clients, transformant Kanban en Scrumban.

Selon les cas, il est parfois préférable de mettre en place de petits groupes de travail pluridisciplinaires, d'autres fois il est préférable de séparer l'équipe informatique en zones larges et spécifiques. Ce qui est sûr et je suis très clair à ce sujet, c'est que Kanban est une méthodologie vraiment agile, très puissante et très productive.

Pourquoi accepter des tâches basées sur des descriptions et non basées sur des mots ?

Lors de l'attribution et de l'accord sur les tâches à résoudre au cours de la journée, ils doivent avoir une description bien détaillée, et ne pas s'entendre sur eux par le bouche à oreille car il est très fréquent que les gens oublient ce qu'ils ont convenu lors des réunions précédentes. De plus, dans le cas où de mauvaises décisions ont été prises concernant les objectifs des tâches, ces descriptions peuvent être utilisées pour soutenir le travail effectué et les décisions qui ont été prises ensemble.

Considérez tous les points de vue et n'assumez pas votre propre véracité

Les informations et les données sont essentielles à la planification des tâches. Plus vous disposez de données et d'informations, plus vous planifierez et fixerez les objectifs avec précision, pour cette raison, il est important d'organiser de courtes réunions avec différents membres de l'équipe, et même de différents domaines, utilisateurs et clients, afin de savoir leurs points de vue et parviennent à trouver une convergence entre toutes les idées. De cette façon, il est possible de détecter des problèmes qui n'ont pas été pris en compte et qui pourraient conduire à des problèmes majeurs à l'avenir.