Chapitre 3
Exécution des tests

1. Objectif

L'objectif de l'exécution des tests est de dérouler les cas de tests préparés pendant la phase en amont de conception et de préparation. Il s'agit de vérifier que le résultat obtenu dans le logiciel ou l'application est conforme au résultat attendu défini dans le cas de test

Lorsque le résultat obtenu est différent du résultat attendu, les écarts sont analysés et, si l'anomalie est avérée, un ticket anomalie doit être créé pour tracer et suivre le traitement de l'anomalie.

La phase d'exécution est une étape clé, attendue par les équipes projet, car c'est à ce moment que l'application est confrontée à la réalité. Les résultats des tests permettent de valider son bon fonctionnement ou de révéler des anomalies. Ces dernières sont ensuite suivies et traitées à l'aide d'outils tels que HP ALM, Jira ou Squash.

2. Campagne de tests

Préparation

Une campagne de tests est un ensemble de tests, sélectionnés et regroupés pour être exécutés sur une période donnée pour une version de l'application.

Chaque campagne de tests a des exigences en termes d'objectifs et en termes de délai ou de durée. L'objectif d'une campagne de tests peut être :

– Tester les nouvelles fonctionnalités du logiciel.
– Tester la non-régression suite à une nouvelle mise à jour ou évolution du logiciel.
– Tester le bon fonctionnement du logiciel suite à une migration de serveur de données.

En fonction de l'objectif de la campagne de tests, le testeur va venir « piocher » dans le référentiel ou le patrimoine de tests préparés. Les tests à intégrer correspondent à la préparation de la campagne de tests.

Avant de démarrer l'exécution des tests, il faut préparer une campagne de tests.

Des outils de gestion de tests comme HP ALM et Jira Xray permettent de créer et gérer les campagnes de tests et de suivre les résultats des tests.

Le testeur est responsable de la création et de l'organisation de sa campagne de tests, le chef de projet test peut approuver si le périmètre de test est complet et si les priorités respectées.

La préparation de la campagne de tests est une activité organisationnelle essentielle. Elle permet de suivre les résultats des tests pour une version donnée de l'application et un objectif précis, tout en assurant le suivi de l'évolution de la qualité du logiciel d'une version à l'autre, grâce à l'historisation des campagnes et de leurs résultats.

L'exécution de tests se fait sur l'environnement de test dédié. S'assurer de la disponibilité et de la bonne configuration de l'environnement de test fait partie de l'étape préparatoire de l'exécution des tests.

Exécution

L'exécution d'une campagne de tests consiste à dérouler un à un les cas de test prévus, en enregistrant progressivement les résultats d'exécution de chaque test, voire de chaque étape lorsqu'ils sont détaillés.

Voici le résultat que peut avoir un test pendant à son exécution :

- Statut « Ok » ou « Passant » selon l'outillage et la nomination choisis : lorsque le résultat obtenu dans l'application est conforme au résultat attendu.
- Statut « KO » ou « En échec » : lorsque le résultat obtenu dans l'application est différent du résultat attendu.
- Statut « Bloqué » : lorsque l'exécution d'un test est empêchée par une anomalie existante, et en raison de dépendances, peut bloquer et affecter d'autres tests.

L'exécution de la campagne de tests comprend également la remontée des écarts identifiés sous forme d'anomalies, le suivi de leur traitement, ainsi que le re-jeu des tests en échec une fois les anomalies corrigées.

Nous détaillerons dans le chapitre suivant la gestion des anomalies qui est une partie intégrante de l'exécution des tests.

3. Gestion des anomalies

3.1 Objectifs

Nous avons vu dans les chapitres précédents qu'une anomalie est un écart entre un résultat attendu et un résultat obtenu détecté sur les environnements de test. Une anomalie est aussi appelée bug ou défaut, on utilise plus souvent le terme anomalie.

Nous appelons les anomalies détectées en production des incidents.

La gestion des anomalies est une étape importante dans le processus de test et de développement. La bonne gestion du portefeuille des anomalies ou ce qu'on appelle aussi backlog des anomalies doit se faire via une méthodologie et un processus bien défini.

La gestion des anomalies commence depuis la détection de l'anomalie jusqu'à son traitement et sa clôture. Une anomalie demeure active tant qu'elle n'est pas corrigée, validée et clôturée.

C'est pour cela qu'une bonne gestion des anomalies tant sur le plan opérationnel que managérial est un facteur clé pour garantir une bonne qualité des livraisons et un pilotage avec des indicateurs fiables et efficaces.

Hormis le respect des critères d'acceptance et la satisfaction des utilisateurs, qui reste l'objectif final et collectif, une bonne gestion des anomalies permet la détection des actions correctives et aussi préventives dans le cadre d'une amélioration continue des livrables, des processus et des bonnes pratiques de tests et de développement afin de capitaliser d'une livraison à une autre.

Une bonne gestion des anomalies facilite également l'organisation et la priorisation des travaux des testeurs et développeurs et facilite surtout la communication et la collaboration entre ces deux équipes.

3.2 Processus de traitement des anomalies

Comme évoqué dans la section précédente, la gestion d'une anomalie commence depuis sa détection jusqu'à sa fermeture. Entre les deux états, l'anomalie passe par différentes étapes : c'est ce que l'on appelle le cycle de vie de l'anomalie, en anglais *workflow* de l'anomalie.

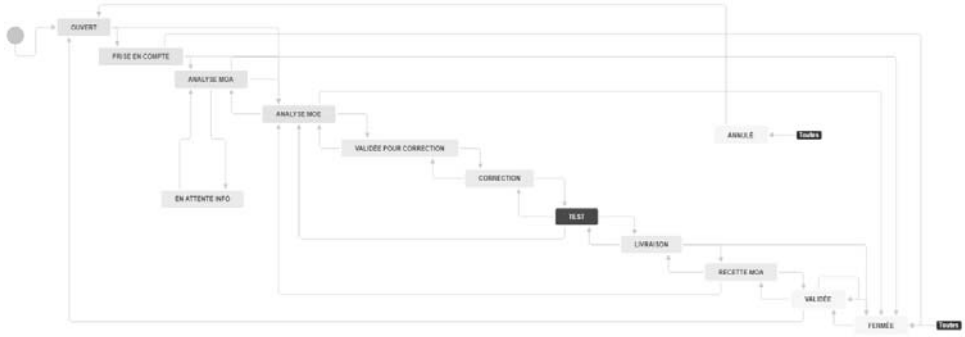
Le cycle de vie d'une anomalie est défini par chaque organisation et formalisé dans la stratégie de tests et parfois plus en amont dans le document PAQ (plan d'assurance qualité).

Le cycle de vie d'une anomalie peut varier en fonction de l'environnement dans lequel elle a été détectée.

Remarque

Une anomalie détectée sur l'environnement de validation aura un cycle de vie avec plus ou moins d'étapes qu'une anomalie détectée sur l'environnement recette utilisateur.

Exemple de workflow



Voici les principales étapes du cycle de vie d'une anomalie.

3.2.1 Détection de l'anomalie

L'anomalie une fois détectée doit être signalée par le testeur au développeur. Une fiche anomalie doit alors être créée en respectant les bonnes pratiques de signalement d'une anomalie. Selon l'outillage utilisé, des champs obligatoires doivent être renseignés pour que la fiche anomalie soit exploitable par l'équipe de développement.

Voici les principales propriétés d'une anomalie :

- **Le titre** : le titre de l'anomalie doit être clair, signifiant et intégrer le nom de la fonctionnalité en échec pour faciliter l'affectation de l'anomalie au bon développeur, c'est-à-dire au développeur ayant travaillé sur l'implémentation de la fonctionnalité concernée ou au développeur maîtrisant le domaine fonctionnel impacté.
- **Les étapes ou le scénario de reproduction** : il s'agit de renseigner toutes les étapes du test ayant permis de détecter l'anomalie. Il faut que les étapes soient complètes et claires en mentionnant, pour chaque étape, les données de tests utilisées ainsi que le résultat attendu et le résultat obtenu. Le développeur doit avoir toutes les informations nécessaires pour qu'il puisse reproduire l'anomalie sur son environnement de développement.

Un manquement dans les étapes de reproduction entraînera un temps supplémentaire d'aller-retour entre le testeur et le développeur ou un rejet de l'anomalie par l'équipe développement. Dans les deux cas, il en résulte un délai supplémentaire évitable pour le traitement de l'anomalie ainsi qu'un temps consommé par le testeur au détriment d'autres tâches.

– **La sévérité** : la sévérité d'une anomalie est une mesure de l'impact qu'une anomalie a sur le fonctionnement d'un système ou d'un processus. Elle aide à déterminer la gravité et l'importance de sa résolution.

Les critères communs pour évaluer la sévérité d'une anomalie sont :

– **Impact sur le fonctionnement** : est-ce que l'anomalie empêche le système de fonctionner correctement ou le rend inexploitable ?
– **Pertes financières** : quel est le coût associé à l'anomalie, en matière de pertes financières directes ou indirectes ?
– **Sécurité** : l'anomalie présente-t-elle un risque pour la sécurité des utilisateurs ou du personnel ?
– **Réglementation et conformité** : l'anomalie entraîne-t-elle des violations de normes ou de régulations qui pourraient entraîner des sanctions ?
– **Effet sur les utilisateurs** : quel est l'impact sur l'expérience utilisateur ? L'anomalie cause-t-elle des désagréments mineurs ou des interruptions majeures ?

Voici des exemples de catégories de sévérité d'une anomalie souvent utilisées :

– **Bloquante** : une anomalie est dite bloquante si elle est liée à un processus métier critique, c'est-à-dire que, si elle survient en production, l'utilisateur sera bloqué dans ses fonctions, ou que l'utilisation du système sera bloquée. Une anomalie peut aussi être taguée comme bloquante si elle bloque l'avancement des tests avec les testeurs bloqués dans leurs travaux.
– **Majeure** : une anomalie est dite majeure si elle est liée à une fonctionnalité majeure et que le système peut continuer à fonctionner.
– **Mineure** : une anomalie est dite mineure lorsqu'elle cause des désagréments mineurs sans impacter ou bloquer le fonctionnement de la fonctionnalité.

Remarque

La définition de chaque sévérité est propre à chaque organisation. Cependant, il est important que la définition soit formalisée et partagée avec l'ensemble des parties prenantes.

– **Priorité** : nous avons vu précédemment la notion de sévérité, qui est l'évaluation de la gravité de l'anomalie sur le système et les utilisateurs. La priorité va venir déterminer l'urgence avec laquelle l'anomalie doit être traitée. Il arrive qu'on ait des anomalies avec la même sévérité, mais de priorité différente. La combinaison des deux va permettre de mieux prioriser le traitement des anomalies.

Prenons un exemple d'application pour mieux comprendre :

- Anomalie 1 : une faille de sécurité critique dans un logiciel bancaire.

Sévérité : bloquante (risque de pertes financières massives et impact sur la sécurité).

Priorité : maximale.

- Anomalie 2 : un bug graphique mineur dans une application de gestion de projet.

Sévérité : mineure (impact visuel mineur, aucune perte fonctionnelle).

Priorité : basse

En évaluant le couple sévérité/priorité, les équipes de développement et de maintenance peuvent prioriser efficacement les anomalies à corriger, en tenant compte de leur gravité et de leur urgence.

La priorité d'une anomalie peut-être évaluée en fonction des éléments suivants :

– **Version** : il s'agit de la version applicative du logiciel.

  Chaque nouvelle livraison équivaut à une nouvelle version. Il est important de maintenir la bonne pratique en développement d'incrémenter les versions applicatives à chaque nouvelle livraison. Dans la fiche anomalie, on utilise deux types de versions. La version dans laquelle l'anomalie a été détectée et la version dans laquelle l'anomalie a été corrigée et validée.

  Cette information permet de mieux maîtriser le contenu de chaque livraison. Dans le cas d'un blocage, il sera plus simple d'identifier la livraison ayant généré le blocage et de pouvoir effectuer un retour arrière, restaurer l'application dans la version antérieure et assurer la continuité de service. Les corrections pourront se poursuivre alors dans un autre environnement, notamment celui du développement sans impact.

Prenons l'exemple d'une anomalie détectée dans une version V1, corrigée dans une version V2 et réapparue dans une version V3. Si le versioning est bien configuré et son renseignement respecté dans la fiche anomalie, on peut facilement constater qu'il s'agit d'une régression et évaluer ensuite la cause racine pour mieux prévenir et anticiper les régressions dans les versions futures.

– **Environnement** : il s'agit de l'environnement dans lequel l'anomalie a été détectée. Cette information est importante pour que le développeur puisse reproduire l'anomalie dans les mêmes conditions que celles du testeur. Le correctif doit être livré sur l'environnement de détection de l'anomalie et, si validé, il doit être propagé par le développeur sur les autres environnements.

– **Rapporteur** : il s'agit de renseigner le nom du testeur qui a détecté l'anomalie. Cela permettra au développeur de prendre contact directement avec le testeur en cas de question ainsi que de le notifier une fois l'anomalie corrigée.

– **Responsable** : il s'agit de renseigner le nom du développeur ou du point d'entrée de l'équipe développement en fonction du processus défini.

– **Commentaire** : dans la majorité des outils de gestion des anomalies disponibles sur le marché, il existe un champ commentaire dans la fiche anomalie.

  Il est recommandé d'utiliser ce champ pour tracer les échanges avec les développeurs ou les autres intervenants dans le traitement de l'anomalie.

Il est aussi recommandé de demander au développeur, pour chaque correction, de renseigner dans ce champ la cause identifiée et le résultat de son analyse pour que cela serve de base pour les prochaines corrections.

3.2.2 Assignation de l'anomalie

L'assignation de l'anomalie c'est son affectation à la personne ou à l'équipe en charge de son traitement. C'est le processus de traitement des anomalies qui définit à qui affecter l'anomalie, et dans quel cas, ainsi que l'état ou le statut de l'anomalie associé.

Une anomalie liée au code peut être assignée au développeur directement, si le testeur connaît le nom du développeur ayant travaillé sur l'implémentation de la fonctionnalité impactée. Sinon, dans d'autres cas, l'anomalie est assignée à un point d'entrée identifié qui peut être aussi le responsable de l'équipe développement. C'est lui qui se charge de répartir les anomalies en fonction des compétences et de la disponibilité de chaque développeur.

Si l'anomalie est de type documentaire par exemple, elle est assignée à l'équipe des *business analysts*.

Remarque

L'affectation d'une anomalie se fait en fonction de sa typologie et du processus de traitement des anomalies définies.

3.2.3 Traitement de l'anomalie

Le traitement de l'anomalie consiste à la prise de connaissance de son contenu par l'équipe concernée.

Si c'est le développeur qui prend connaissance de l'anomalie, il va tout d'abord essayer de la reproduire sur son environnement de développement en local. D'où l'importance, comme soulevé dans la section précédente, de bien décrire les étapes du scénario ayant permis de détecter l'anomalie. Si l'anomalie est avérée, il procédera à sa correction. Si l'anomalie n'est pas reproductible en local, le développeur fera une analyse plus poussée ou retournera l'anomalie au testeur.

Chapitre 3
Versant métier

1. Introduction

La connaissance du métier est un des versants les plus forts dans le test, à tel point qu'une personne qui n'a aucune vision sur le test saura trouver des anomalies à forte valeur ajoutée tandis qu'un testeur professionnel sans connaissance métier saura s'organiser, mais il devra partir à la pêche aux informations pour commencer à savoir ce qu'il faut chercher…

Ce chapitre va répertorier quelques pratiques qui viendront vous aider dans l'obtention d'un meilleur produit par des pratiques qui vont permettre la maîtrise du métier.

2. Quelques techniques d'idéation

Que ce soit lors d'un PI planning ou d'un Sprint Refinement, avant d'arriver au sein d'une équipe de réalisation, les besoins émergent par certains méandres de l'organisation. Cette section montre quelques techniques qui peuvent être utilisées dans des équipes liées au management de produit tel qu'une équipe marketing.

Dans le cadre du test, l'intérêt porté sur la fourniture des besoins tient à deux points :

– elles font partie de la base de test [ISTQB 2023] et sont donc la **matière première des tests**, et en tant que tels la qualité des tests s'en ressentira ;
– parce qu'elles font partie du socle qui définit le produit et constitue une base **tangible de la connaissance du produit** (voir chapitre Facteurs de succès).

C'est pourquoi, dans une approche du « tester tôt », nous allons voir comment peut se passer la formalisation des besoins afin d'être capable de traiter les défauts au plus près de leurs racines, sachant que selon Boris Beizer, cette partie provoque environ **8 % des anomalies** d'un produit [Beizer 1990][Laporte 2017]. Cependant, la rentabilité des actions préventives et correctives liées aux tests sur l'idéation est évidente, car elles sont situées tôt dans la génération du produit.

2.1 Design Thinking

Le Design Thinking, démarche inspirée par la conception que l'on peut traduire par « *Pensée par le Design* », trouve ses racines dans les années 1970 à 90 avec les travaux de McKim [McKim 1972] [McKim 1980] et Faste [Faste 1993]. C'est une approche de l'innovation basée sur l'observation directe de ce que veulent les utilisateurs et le marché. Elle suit les phases suivantes [Lewrick 2018] :

- **comprendre** notamment les types d'utilisateurs ;
- **observer** les types d'utilisateurs ;
- **définir un point de vue** sur les points relevés par les types d'utilisateurs ;
- **faire une idéation** créative des points de vue ;
- **développer des prototypes** autour des idéations ;
- **tester** les prototypes avec les utilisateurs ;
- **mener une réflexion** à partir des feedbacks utilisateurs.

Sur chaque phase, le Design Thinking propose tout un panel d'outils tels que la notion de persona (voir le chapitre Versant industriel du test à la section Technique de la persona), l'*Empathy map* (voir ci-dessous), le *Test A/B* (voir le chapitre Versant technique du test à la section A/B Testing) et bien d'autres que vous pourrez trouver notamment chez Lewrick, Lockwood [Lockwood 2009] ou Moote [Moote 2013].

Empathy mapping

Quand on définit un service, on avance un peu dans le noir et on n'imagine pas comment un client peut penser ou agir dans son contexte d'utilisation. Le premier pas vers le métier peut être une « carte d'empathie » (en anglais, « Empathy map ») et s'inscrit dans la phase d'observation du Design Thinking.

Cette notion, popularisée par Alex Osterwalder [Osterwalder 2013] et déclinée notamment par [Boukobza 2017], nous vient de Dave Gray [Gray 2009]. Gray proposait de découvrir un maximum de choses sur les besoins de « l'autre » et se glisser dans la peau des clients via un atelier qui les invite à s'exprimer.

Dresser une empathy map avec ses clients aide à [VanZandt 2023] :

- *définir des stéréotypes de clients (voir la notion de persona – voir chapitre Versant industriel du test) ;*
- *créer un produit dans l'état d'esprit de ses clients ;*
- *éviter d'avoir à mettre en place de lourdes études marketing pour tenter de les connaître ;*
- *identifier les risques liés à votre produit à partir du terrain (voir la notion de Gemba ci-dessous).*

Pour y parvenir, on part du schéma ci-dessous, composé de six parties sur chacune desquelles des post-it peuvent être posés de façon séquentielle.

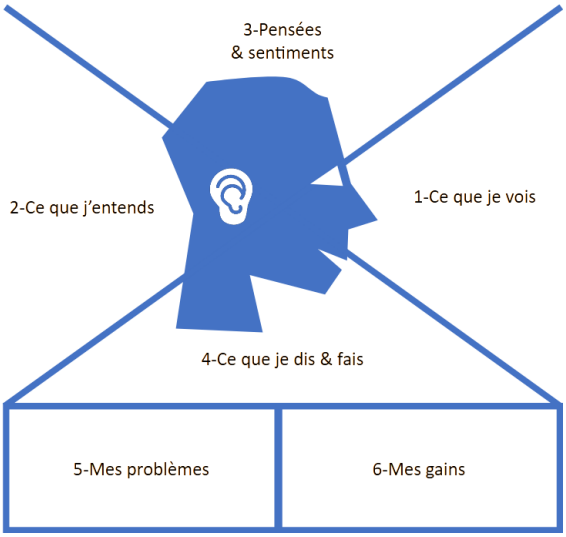


Figure III-1 : Schéma de l'empathy mapping

1. Ce que je vois : chaque type de client perçoit le produit dans son contexte. Il s'agit que chacun décrive ce qu'il voit autour de lui ; qu'il donne ses alliés et amis, les détracteurs et ennemis, les offres auxquelles il est exposé et les problèmes qu'il rencontre.

2. Ce que j'entends : les clients sont amenés à dire ce que disent leurs amis, les supports et les canaux qui pourraient les influencer, ce qui les influence réellement et comment.

3. Pensées et sentiments : les clients doivent préciser ce qui est vraiment important pour eux, ce qui les émeut et qui parfois les tient éveillés la nuit, et donc leurs rêves et aspirations.

4. Ce que je dis et fais : il s'agit ici d'identifier ce qu'ils peuvent dire aux autres, en public, comment ils se comportent en public.

5. Mes problèmes : ici, les clients vous confient leurs plus grandes frustrations, quels sont les obstacles qu'ils rencontrent entre ce qu'ils recherchent et leur situation actuelle ; quels sont les risques auxquels ils sont confrontés.

6. Mes gains : que souhaitent réellement vos clients ? Comment vont-ils mesurer le succès ? Quelles stratégies vont-ils employer pour atteindre leurs objectifs ?

Avec de telles informations, n'importe quel testeur devrait pouvoir tirer suffisamment de matière pour imaginer ce qu'il va pouvoir réaliser comme test sur le produit, au-delà d'une simple conformité à des spécifications.

Session Gemba

Comme le rappelle le 6e principe du test : « Les tests dépendent du contexte ». Au-delà des considérations internes du test telles que l'environnement technique sur lequel un test sera exécuté, le contexte, dans lequel un produit est utilisé, a un impact considérable sur sa perception et son usage par les utilisateurs.

Une fois de plus, le lean nous propose un mot japonais 現場, « Gemba » (ou « Genba ») :

- *現 est composé des parties 王 (le roi, le règne) et 見 (voir, espoirs, chances, idée, opinion, regarder, visible) pour parler du présent dans le monde réel.*
- *場 dont les parties (一·土·日·勿) peuvent se comprendre par « un lieu, un jour, tout en un coup » et se traduit par « l'endroit ».*

Genba est donc le « lieu réel », mais aussi la « scène de crime ». C'est là où tout se joue au quotidien. C'est la clé de la connaissance tacite [Nonaka 1998] et du non-dit car admis dans la culture du contexte dans lequel tout se passe [Busch 2004] voire l'indicible. Ainsi, pour comprendre les aspects tacites, il est nécessaire d'aller sur le terrain pour comprendre, plutôt que d'analyser derrière son bureau des artéfacts, certes tangibles, mais exempts de ces non-dits.

Ainsi, l'idée d'une session Gemba (en anglais on dit « Gemba walk », « promenade sur place ») est de s'intéresser aux personnes et non aux choses. Classiquement, une session Gemba est utilisée pour comprendre un goulot d'étranglement, un gaspillage ou un problème particulier [Womack 2011].

Le double diamant de Bánáthy

En 1996, Béla H. Bánáthy a proposé une approche du design basé sur un principe simple de divergence-convergence appliqué dans un premier temps à la création de la vision puis à sa réalisation ; ce qu'il a illustré par deux losanges, en anglais « diamond » qui s'enchaînent [Bánáthy 1996] :

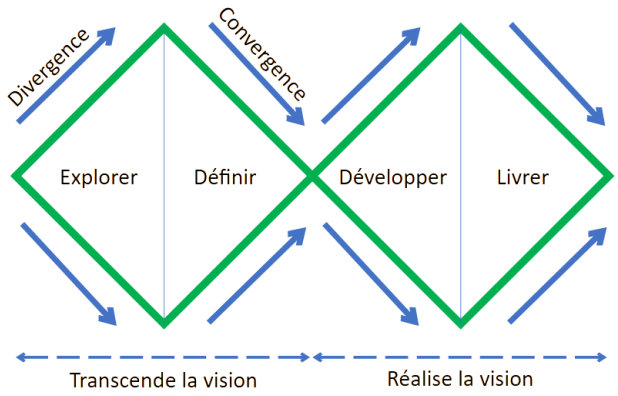


Figure III-2 : Le double diamant de Bánáthy – illustration [Moustier 2020]

Cette approche du design vient généraliser la démarche du design thinking dans sa succession de phases qui d'un point de départ vient explorer les possibilités et retenir une solution.

Ce principe de divergence-convergence permet d'éviter de partir tête baissée sur une idée qui peut mener à un échec si l'ensemble des choix n'a pas été exploré au préalable. Nous pouvons retrouver cette approche dans ce que le CMMi appelait « Technical Solutions » en invitant l'équipe à proposer des solutions techniques alternatives [CMMI 2010].

2.2 Lean Start-up

Lorsque des études donnent l'hypothèse d'une fonctionnalité gagnante, c'est-à-dire qu'elle obtiendra du succès auprès des utilisateurs, il faut alors pouvoir tester que cette hypothèse est la bonne.

Pour cela, l'équipe de réalisation va tenter de réaliser le minimum de code pour que cette hypothèse prenne vie afin que les utilisateurs puissent l'utiliser et faire leurs premiers retours, c'est-à-dire un MVP (voir chapitre Quelques rappels - section Point de vue du modèle Spotify). À partir du feedback, il faudra prendre la décision de continuer dans cette voie ou abandonner ce qui a été fait, sans remords.

Cette approche a été imaginée par Eric Ries en 2012 à partir de son expérience d'entrepreneur. Il l'a baptisée *Lean-Start-up*. Elle est basée sur le paradigme « *Fabriquer-Mesurer-Apprendre* » [Ries 2012].

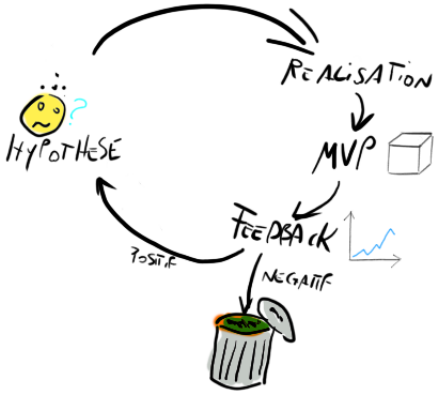


Figure III-3 : Le cycle du Lean Start-up – si le retour du terrain n'est pas bon, on jette, sinon on peut persévérer

Vous noterez que le Lean Start-up induit deux choses :

– être capable de mesurer la satisfaction de l'hypothèse ;
– être capable de jeter ce qui a été fait...

Leading Indicators – les indicateurs clés de succès

Chez SAFe, lorsqu'une initiative est générée et que ce récit utilisateur doit s'inscrire sur la durée, il prend la forme d'une épopée, ou en anglais, « epic ». Afin de savoir si cette initiative vaut la peine qu'on y investisse de l'énergie, SAFe propose de mettre dans la description de l'epic, une liste d'indicateurs de sastisfaction de l'initiative [SAFe 2023-10].

Par exemple, si on imagine un site de e-commerce qui vendrait des chaussures, plutôt que de louer ou acheter un entrepôt pour stocker ses produits et fabriquer un site avec une équipe complète d'ingénieurs, on peut imaginer conclure un marché avec des boutiques et faire une simple page avec un catalogue de photos prises dans ces magasins ; ce sera votre MVP. Cet exemple de MVP est celui de la société Zappos.com créée par Nick Swinmum en 1999 à une époque où personne ne savait si les gens allaient accepter d'acheter des chaussures sur Internet [Ries 2012].

Dans le même temps, pour savoir si votre site a du succès, vous allez par exemple mesurer l'AARRR (voir la section Qualité de service au chapitre Versant techologique du test). Ces indicateurs doivent être intégrés à votre MVP si vous voulez pouvoir vérifier votre hypothèse.

Ainsi, la différence entre un MVP qui serait un simple prototype et le MVP proposé par le Lean Start-up est que le produit doit

- *être* ***commercialisable*** ;
- *contenir le* ***code nécessaire à la mesure*** *de vos Leading Indicators.*

Cette notion de feedback est une forme de test qui est réalisée, non pas par l'équipe de réalisation, mais par les clients. On retrouvera cette vision du test dans le chapitre Versant technique du test - section Stratégie Shift Right.

Aversion à la perte

Ce qu'ont montré Amos Tversky et Daniel Kahneman en 1991 c'est que nous sommes tous enclins à penser qu'une chose dont vous devez vous séparer a plus de valeur que le prix auquel vous l'avez achetée [Tversky 1991]. Lorsqu'on a passé du temps à faire quelque chose, il est toujours difficile de s'en séparer ; encore plus lorsqu'il s'agit de la jeter !

MVP/MMF/MMR/MMP

Depuis 2012, la notion de MVP s'est bien répandue et a été peaufinée. En effet, lorsqu'il s'agit d'un nouveau produit, il est évident qu'un prototype commercialisable et instrumenté est approprié. En revanche, avant d'en arriver là, il peut être pertinent d'avoir des sortes de MVP dédiés à certains contextes ; et lorsqu'il s'agit d'un produit existant, est-ce qu'on peut toujours parler de MVP ?

Pour cela, certains auteurs ont étendu cette notion à différents contextes. Par exemple, Don McGreal identifie plusieurs types de MVP avant la mise sur le marché . Il répertorie notamment un MVP de promotion, de test, de « Landing page », ou simplement conceptuel (basé sur des écrans en papier ou en wireframe) [McGreal 2018] [Moustier 2020].

Par ailleurs, lorsque le produit existe déjà et qu'il s'agit d'avoir le concept du MVP appliqué à une nouvelle fonctionnalité ajoutée ou une nouvelle release, nous parlerons alors de :

- *MMF : « **Minimum Marketable Feature** » pour définir les éléments fonctionnels suffisants pour que la fonctionnalité soit mise sur le marché [Denne 2003] ;*
- *MMR : « **Minimum Marketable Release** » pour définir les éléments suffisants pour qu'une nouvelle version soit mise sur le marché [Ambler 2012] ;*
- *MMP : « **Minimum Marketable Product** » : la première version d'une MMR [Mendes 2023].*

2.3 Event Storming

Comme le propose Sutherland, une idée clé de l'agilité est d'atteindre une organisation de type C [Sutherland 2011] afin de « tout faire en même temps ».

Pour cela, Brandolini a proposé en 2019 une approche nommée *Event Storming* [Brandolini 2019].

Cette approche consiste en un atelier qui réunit dans une même salle, tous les acteurs du projet :

- le métier ;
- les architectes ;
- les développeurs.

L'idée est de faire plancher tout le monde en même temps sur les besoins métier et la technique afin d'identifier un MVP avec des post-it et des markers de couleurs différentes pour modéliser un processus métier :

- **Phase 1** : 2 à 3 heures
  - Le métier place les **événements métier** dans le sens chronologique avec des notes oranges – ex. « *l'utilisateur lance la commande X* », « *le serveur Y lance une action Z* », « *à la date D, tel batch est lancé* ».
  - La technique place des **remarques** à côté des événements avec des notes violettes telles que « *pas clair* », « *à quoi correspond la donnée X ?* », « *que se passe-t-il lorsque...* », *etc.*
  - Le métier indique sur des notes les **termes** avec une autre couleur.
- **Phase 2** : décoration des événements
  - Les événements sont décorés de post-it bleus qui représentent les **commandes**.
  - Les **types d'utilisateurs** sont identifiés sur les commandes par des notes bleues.
  - Les **données** impliquées sont ajoutées avec des notes vertes.
  - Les **règles métier** sont rajoutées avec des notes mauves.
- **Phase 3** : création des « contextes bornés »
  - Les commandes et événements d'un même contexte sont étiquetés avec un post-it jaune pâle qui porte le nom du contexte borné.