Yest®
by Smartesting

Test design

Le guide complet pour des tests performants

www.smartesting.com

Auteur(e)s

Anne Kramer

Formatrice IA

Depuis 2006, Anne n'a cessé de promouvoir le MBT en tant que cheffe de projet, consultante, formatrice et auteure. Elle rejoint Smartesting en 2022 et explore un nouvel univers : les Large Language Models (LLMs) et l'IA Générative.
Forte de l'expertise acquise aux côtés des spécialistes IA de Smartesting, elle partage aujourd'hui ce savoir à travers notre formation sur l'IA générative.

Arnaud Bouzy

Product Manager

Arnaud Bouzy a commencé sa carrière comme développeur de logiciels de CAO chez Matra Datavision puis avant-vendeur chez Rational et IBM, où il a travaillé sur la modélisation et le test dans les systèmes embarqués. Il élargit son champ d'action chez Smartesting en tant que Product Manager, porteur de la vision produit de notre entreprise.

Elodie Bernard

Product Manager Yest

Docteure en informatique, spécialisée en Model-Based Testing, Elodie évolue depuis 8 ans dans le domaine du test logiciel. Elle accompagne de grands comptes à travers des missions de conseil et d'expertise, couvrant tout le cycle de test, de la conception à l'automatisation.

Michel Guez

CEO de Smartesting

Après plus de 20 ans dans le test logiciel et l'ingénierie de la qualité, il a vu les pratiques évoluer et le marché du test se transformer. Passionné par l'innovation, il conçoit des solutions qui simplifient et optimisent le travail des équipes QA. Son objectif et celui de son équipe : offrir des outils performants et efficaces pour relever les défis d'aujourd'hui et de demain.

Le test design

Sommaire

Introduction

Bienvenue dans l'univers du Test Design !

Le **test design** est une discipline clé du développement logiciel, où rigueur et innovation se rencontrent pour garantir des applications fiables et performantes. À travers cet ebook, nous vous invitons à plonger au cœur de cette approche stratégique : comprendre ses **fondamentaux**, identifier ses **acteurs clés** et relever les **défis** qu'elle permet de surmonter.

💡 **Qui sommes-nous ?**
Chez **Smartesting**, éditeur français spécialisé dans les solutions de test logiciel, nous repoussons les limites du **Quality Assurance (QA)** avec des outils innovants intégrant l'intelligence artificielle. Depuis plus de 5 ans, nous investissons en **technologie IA** pour transformer et optimiser les processus de test.

🚀 **Nos solutions phares :**
Yest, une plateforme de conception visuelle de test qui révolutionne la création et l'implémentation des tests :

- **Conception graphique intuitive et collaborative.**
- **Accélération de l'implémentation des tests jusqu'à 40 %.**
- **Optimisation de l'automatisation,** avec une forte implication des testeurs manuels.
- **IA intégrée** pour reprendre et améliorer les tests existants.

Gravity, une solution qui optimise le ciblage des tests end-to-end pour une exécution plus rapide et plus fiable.

Dans cet **ebook**, **explorez avec nous la conception de test, première étape essentielle du cycle de vie des tests logiciels.** Préparez-vous à repenser votre approche du QA et à découvrir comment Smartesting vous aide à améliorer la qualité de vos logiciels, tout en gagnant en efficacité et en agilité.

Bonne lecture !

I. Découvrir le test design

1.1 Qu'est-ce que la conception de test ?

La production de cas de test repose sur **trois activités étroitement liées** : l'**analyse**, la **conception** et l'**implémentation** des tests.

Lors de l'**analyse**, les testeurs et testeuses s'approprient le sujet, cherchent à en **comprendre les détails** et posent de nombreuses questions au product owner, à l'analyste métier ou à tout autre personne maîtrisant les besoins. Cette phase s'apparente à l'élicitation des exigences, mais avec un niveau de détail différent. Elle permet également d'**examiner les comportements attendus en cas d'utilisation imprévue**, d'**anticiper les erreurs possibles** et d'**explorer les limites** du système.

La **conception des tests** consiste ensuite à déterminer **comment mettre le système à l'épreuve.** C'est à ce moment que les méthodes de conception de test, présentées plus loin dans ce livre blanc, entrent en jeu. Ces **approches méthodiques** permettent de poser des questions précises, qui émergent naturellement lors de l'analyse des tests.

Exemple :
Si nous souhaitons tester une fourchette de valeurs, l'analyse des limites nous indique qu'il faut également vérifier les valeurs non valables. **Mais que se passe-t-il si l'on fait une demande de crédit pour un montant négatif ?**

Ce type de question survient précisément au cours de la conception des tests, rendant difficile la distinction entre **analyse et conception.** On peut ainsi considérer ces étapes comme une continuité de l'analyse des exigences. C'est aussi pourquoi **la conception des tests concerne toute l'équipe.**

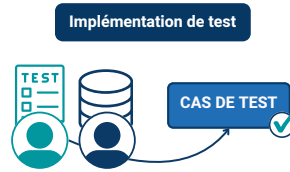
L'implémentation des tests

L'implémentation intervient en **dernier**, **avant l'exécution des tests**, et s'intègre aux autres activités définies par l'ISTQB, comme la planification et la clôture. À cette étape, **les cas de test sont affinés** et **documentés** de manière plus exhaustive, avec des données précises et, si nécessaire, des étapes détaillées.

Exemple :
Lors de la conception des tests, je rédige un **ensemble de scénarios** permettant de vérifier les règles d'acceptation d'un crédit. J'indique, par exemple, que pour être accordé, le prêt ne doit pas dépasser un taux d'endettement de 33 %. Si ce seuil est dépassé, le prêt est refusé.

Pendant l'implémentation, je complète ce test en **intégrant des données concrètes :** salaires, montant du prêt demandé… afin de vérifier le respect du taux d'endettement.
Je peux aussi ajouter une description détaillée pour permettre à un consultant externe de l'**exécuter manuellement** ou de l'implémenter en **script automatisé.**

À partir de l'analyse des spécifications/user stories, produire des cas de test pour mettre le système à l'épreuve.



Compléter les tests avec des données concrètes, des descriptions pas-à-pas pour l'éxécution des tests.

Généralement, les **entrants à la conception** des tests sont les **spécifications**, **exigences**, et **user stories** devant être vérifiées par les tests. En sortie, un ensemble de cas de tests est produit. Ces cas de test devront être implémentés (entre autres, complétés avec des données) pour permettre leur exécution.

La conception de test : une activité sous-estimée

Souvent, la **conception des tests passe inaperçue** aux yeux des managers, qui la confondent avec l'implémentation.

Cela s'explique par plusieurs raisons :

- La **conception des tests** ne génère pas forcément une **documentation volumineuse.** Dans un outil ALM, un simple cas de test contenant une description succincte de son objectif, de la situation à tester et du résultat attendu peut suffire. **Vu de l'extérieur, cela peut donner l'impression qu'il ne s'agit que d'une ébauche.**

- **L'implémentation des tests** est une **activité lourde et coûteuse**, mobilisant des ressources et entraînant des frais de maintenance élevés. De ce fait, elle capte davantage l'attention des managers, qui cherchent à **optimiser les coûts et l'efficacité.**

Pourtant, la **conception des tests apporte une valeur ajoutée majeure :** elle instaure une compréhension commune entre toutes les parties prenantes et **favorise la collaboration.**
Bien qu'immatérielle, **cette valeur est cruciale pour la réussite du projet.**

Dès que les éléments nécessaires sont disponibles, il est recommandé de démarrer la conception des tests. C'est le principe du **Shift-Left**, qui encourage à **intégrer les activités de test et d'assurance qualité le plus tôt possible** dans le cycle de développement (ISTQB).

Concevoir des tests pour une exécution manuelle ou automatisée

Les objectifs de **la conception et de l'implémentation des tests sont distincts :**

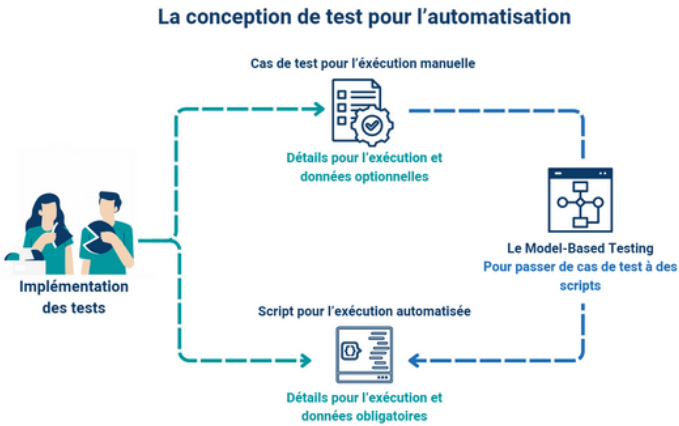
- **La conception vise à définir des tests pertinents** couvrant les règles métier, sans se préoccuper des détails d'exécution. Elle est indépendante du mode d'exécution (manuel ou automatisé).

- **L'implémentation apporte ces détails :** elle précise les données nécessaires pour valider les règles métier établies en phase de conception.

Les tests manuels **doivent inclure suffisamment d'informations** pour leur exécution.
Leur granularité dépend du testeur et il est courant de rédiger des étapes détaillées décrivant les actions à réaliser et les résultats attendus dans un outil de gestion de test.

À l'inverse, les tests automatisés nécessitent des informations précises. **Un script ne peut pas comprendre une consigne vague** comme "demander un prêt dépassant 33 % d'endettement". Il lui faut des valeurs concrètes. Cependant, cela ne signifie pas que ces données doivent être intégrées en dur dans les tests.
Une séparation entre la logique métier et les détails d'implémentation, appelée couche d'adaptation (ISTQB), permet une meilleure flexibilité. Le **keyword-driven testing** est une **approche populaire** reposant sur cette architecture.

Bien que distincts, les tests manuels et automatisés peuvent être liés, ce qui offre plusieurs avantages. Par exemple, le **Model-Based Testing permet d'unifier ces deux approches.**



Et si les tests doivent évoluer ?

Comprendre la **distinction entre conception et implémentation des tests** permet de créer des tests à la fois pertinents et plus faciles à maintenir.

Deux types de modifications peuvent survenir :

1. **Modification ou ajout d'une règle métier**
   → Cela implique une mise à jour de la conception des tests, afin de bien intégrer le changement. L'implémentation ne suit que lorsque la couverture fonctionnelle est assurée.

2. **Modification d'un aspect technique**
   → Dans ce cas, la conception des tests ne change pas forcément. Avec une architecture bien pensée, seule la couche d'adaptation peut être impactée, sans altérer les scripts eux-mêmes.

En résumé, la **conception des tests** est une **étape clé** qui **garantit la qualité et la pertinence** des scénarios de test. Elle assure une couverture exhaustive des exigences, bien au-delà d'un simple suivi formel dans un outil ALM. **Il est donc essentiel de lui accorder l'attention et les ressources qu'elle mérite.**

Conception de test de qualité = qualité globale des applications

Analyse de test

Conception de test

Implémentation de test

Comment l'IA va changer le monde du test ?

Le développement des grands modèles de langage (Large Language Models / LLM) va **révolutionner la conception des tests**. Grâce aux chatbots comme ChatGPT, nous disposons déjà d'assistants intelligents capables de nous délester de tâches répétitives et de **nous inspirer de nouvelles approches.**

Voici quelques exemples de la manière dont l'IA conversationnelle peut contribuer à l'**analyse, conception et mise en œuvre des tests :**

- Obtenir un aperçu rapide des fonctionnalités à tester
- Créer des cas de test
- Identifier des classes d'équivalence
- Formuler des cas de test
- Traduire des cas de test dans un format spécifique (ex. Gherkin)
- Générer des données de test
- Réduire / optimiser les cas de test existants

Cependant, ce n'est que le début. L'IA est en train de s'intégrer progressivement dans les outils de test.

Par exemple, avec **Yest – l'outil de conception visuelle de tests** de Smartesting, il sera bientôt possible d'analyser et d'optimiser les cas de test existants, puis de les convertir en une représentation visuelle.

Cette avancée ouvre aux entreprises des **opportunités inédites pour améliorer leurs processus de test**, notamment en matière de collaboration !

1.2 Qui sont les parties prenantes ?

Comme nous l'avons vu, **la conception de test** est une **tâche centrale** qui mobilise une part significative des efforts de test. Dans le prochain chapitre, nous verrons qu'elle est également cruciale pour la performance globale de l'activité de test.
Ainsi, il est essentiel que la conception de test soit menée sous la supervision et avec la contribution **des acteurs clés** de l'équipe produit :

- **Le chef de projet**
- **Les professionnels du test**
- **Les responsables métier**

Rôle du chef de projet

L'enjeu principal du chef de projet est de **s'assurer que la conception de test démarre au plus tôt.** Pour cela, il organise la réunion des "**3 Amigos**", où **le backlog** est :

- Partagé
- Explicité
- Discuté et challengé

Cette réunion réunit le **représentant des Métiers / Product Owner**, un développeur, et un testeur. Ensuite, **le chef de projet valide la stratégie de test proposée** par le test lead, en gardant à l'esprit que l'objectif est de réduire les risques avant la mise en production, tout en **optimisant le nombre de tests.**

Rôle du professionnel du test

Pour les testeurs, la conception des tests est un moment clé, car elle **justifie toute l'étape d'exécution des tests.**
Des tests non pertinents n'ont aucun intérêt à être exécutés !
Pour concevoir les meilleurs tests, il faut : analyser les entrants fonctionnels (spécifications, exigences, user stories…), identifier les enjeux et les risques, hiérarchiser les scénarios de test.

Il est aussi **primordial de formaliser cette réflexion**, notamment en : dessinant les workflows à tester, identifiant les règles métier clés, définissant les données associées, traçant les liens avec les exigences fonctionnelles.

La conception de test est trop importante pour ne sortir que d'un seul cerveau !

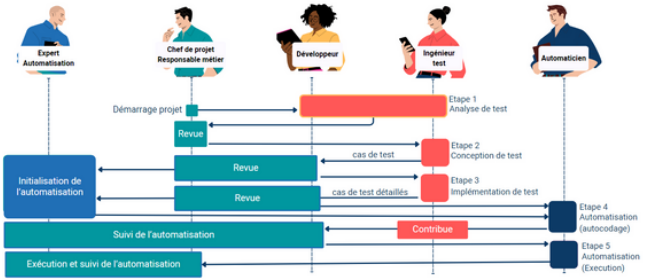
Rôle du responsable métier

Les tests sont un outil stratégique pour les responsables métier, car ils garantissent que les **besoins du marché sont bien implémentés** dans le logiciel.
Ne pas s'impliquer dans la conception des tests revient à s'arrêter à mi-chemin.

Le responsable métier doit donc :

- Effectuer une revue approfondie des tests
- S'impliquer dès le début en challengeant les testeurs
- Discuter et affiner la stratégie de test
- Exiger des artefacts de conception de test plutôt que des tests bruts et implémentés

Exemple d'organisation autour de la conception de test



1.3 Les enjeux auxquels le test design répond

L'implication des **différents acteurs** dans la conception de test est essentielle, car les enjeux sont **majeurs.**
Positionnée très tôt dans le processus de test (et encore plus avec le shift-left), la conception des tests a un impact direct sur toutes les phases suivantes. On devrait même dire qu'elle les justifie entièrement.

Quel sens aurait-il d'exécuter des tests :
❌ Sans prise en compte des risques en production
❌ Sans réflexion sur la couverture optimale ?

Investir dans la conception de test, c'est :
✅ Gagner en maîtrise et en visibilité
✅ Prioriser les tests selon leur criticité
✅ Faire les bons choix en cas de contraintes (temps/budget)
✅ Sélectionner intelligemment les tests à automatiser

L'intelligence du test doit primer sur la quantité de tests. Mais ce n'est pas tout !

La conception de test est une traduction concrète des besoins métier, au même titre que le développement du code. Plus elle est réalisée tôt et avec sérieux, plus les "bugs de spec" sont détectés en amont. Or, **corriger un bug tôt coûte beaucoup moins cher.**

Une conception de test au plus tôt = moins de risques en production

Conception de test

+ de maitrise
+ de visibilité
+ de tests pertinents
+ une meilleur priorisation
+ d'automatisation

Mise en production

- moins de risques
- moins de bugs
- moins de corrections
- moins de coûts

Un levier puissant pour l'agilité. Les réunions des "3 Amigos" permettent de clarifier les besoins et d'aligner toutes les parties prenantes. **Sans collaboration, il n'y a pas d'agilité.**
Plus tard, les techniques de test deviennent trop complexes pour que les Métiers et les développeurs puissent encore apporter une contribution pertinente.

💡**C'est donc lors de la conception des tests que le test trouve toute sa place dans l'équipe agile.**

Un atout pour l'onboarding et la continuité des équipes :

Les équipes connaissent des mouvements de personnel. Un nouveau QA qui rejoint le projet a besoin de monter en compétence rapidement. Les documents de conception de test (surtout les graphiques) lui offrent une **vision claire et rapide** :

- De la logique fonctionnelle de l'application
- Des enjeux de test identifiés par l'équipe

Bien plus efficace que de lire des tests sous forme de scripts !

Un outil pour évaluer la criticité des anomalies :
Tous les bugs n'ont pas le même impact. Grâce à la conception de test, il est possible de :

- Remettre une anomalie dans son contexte
- Évaluer correctement sa criticité business

Enfin, pour les ESN qui réalisent des Tierces Recettes Applicatives (TRA), la conception de test est **le meilleur moment pour :**

- Échanger avec le client
- Éviter les incompréhensions
- Réduire les reworks coûteux

II. Techniques & approches

2.1 Différence entre les techniques de test et les approches de test

Il existe **différentes techniques de test**, dont certaines sont plus connues que d'autres.
Par exemple, la plupart des testeurs connaissent le **partitionnement par équivalence** et l'**analyse des valeurs limites**, mais peu d'entre eux effectueront un jour des tests de réseaux orthogonaux. Les techniques de test fournissent des **instructions à suivre pour obtenir un certain résultat.**
Si vous effectuez un partitionnement par équivalence, vous êtes sûr de couvrir au moins toutes les plages de valeurs distinctes pour les paramètres d'entrée et de sortie.

Les **approches de test** vont plus loin. Elles définissent la manière dont vous mettez en œuvre vos tâches de test. Une approche de test peut également spécifier comment les testeurs communiquent avec les autres parties prenantes et comment ils documentent les résultats.
Le développement piloté par le comportement (BDD) et le **test basé sur le modèle (MBT)** sont des **approches bien connues.**

Dans ce livre blanc, nous examinerons q**uatre techniques** de test et **deux approches** de test :

- **Partitionnement par équivalence**
- **Analyse de la valeur limite**
- **Technique de l'arbre de classification**
- **Test de la table de décision**

Ces quatre techniques sont des **techniques de boîte noire**, ce qui signifie que la conception du test est basée sur les spécifications plutôt que sur la structure interne du système ou du composant testé.

Les **deux approches de test, BDD et MBT,** seront étudiées en détail, en mettant en avant leurs **points communs** et leurs **différences.**

2.2 Techniques de test populaires

Répartition des équivalences - Tester au moins un représentant de chaque plage de valeurs.

Prenons un exemple qui, bien que peu politiquement correct, est facile à comprendre.
La plupart des pays appliquent des règles de protection pour empêcher les mineurs d'abuser de l'alcool.

En Allemagne :

- Si vous avez **moins de 16 ans**, vous ne pouvez acheter **aucun type d'alcool.**
- Entre **16 et 18 ans**, vous êtes **autorisé à acheter et à consommer** des boissons "douces" comme la bière et le vin.
- À **18 ans**, vous êtes considéré comme adulte et pouvez **acheter des alcools forts** et des produits plus dangereux, comme les alcopops.

En **Amérique du Nord**, les **règles sont plus strictes**, mais le principe reste le même :
Il existe des tranches d'âge, et l'âge exact n'a pas d'importance tant qu'il appartient à une tranche définie. Le **résultat attendu repose** uniquement sur ce que l'on appelle **une partition d'équivalence.**

Dans cet exemple, nous avons **deux paramètres d'entrée :**

1. **L'âge**
2. **Le type de boisson**

Mais nous devons également prendre en compte le résultat attendu et nous assurer de tester les deux partitions d'équivalence : **autorisée ou rejetée.**
Un bon test basé sur le partitionnement par équivalence couvre chaque partition avec au moins un cas de test.

ID	Age		Type of drink		Result
	equivalence partition	value	equivalence partition	value	equivalence partition/value
TC_EP1	under 16	10	"soft" stuff	wine	rejected
TC_EP2	16 to 17	17	"soft" stuff	beer	allowed
TC_EP3	16 to 17	17	hard stuff	hard liquor	rejected
TC_EP4	18 or above	65	"soft" stuff	champaigne	allowed

Tableau 1 : Quatre cas de test couvrant toutes les partitions d'équivalence valides (exemple)

Le tableau 1 présente **quatre cas de test** qui couvrent **toutes les partitions d'équivalence** des trois paramètres. Nous avons volontairement introduit quelques variations dans les valeurs concrètes du type de boisson.
En théorie, le **partitionnement par équivalence** ne requiert que **deux valeurs** pour les deux types d'alcool, tant qu'elles couvrent les **deux catégories.** Ainsi, nous aurions pu tester uniquement avec le vin et les alcools forts.

Bonne pratique
Documentez quel cas de test couvre quelle combinaison de partitions d'équivalence afin de garantir une couverture complète. Par exemple, vous pouvez ajouter une description du cas de test ou utiliser des noms de cas de test explicites.

Cependant, un problème demeure : **nous avons négligé les partitions non valides.**
Pour une couverture complète, nous devons inclure plus de tests afin d'évaluer le comportement du système dans des situations inattendues.
Par exemple :

- **Un âge inconnu**
- Une **boisson ne figurant pas** dans les catégories prédéfinies

Dans ces cas, le **résultat attendu** n'est pas toujours clair. **Y aura-t-il un message d'erreur ?**

Si oui, alors nous avons oublié une **partition d'équivalence** pour le **paramètre de sortie.**
Ces valeurs sont probablement **impossibles à saisir via l'interface utilisateur,** c'est pourquoi les partitions non valides sont généralement couvertes lors des tests unitaires.

Le **partitionnement par équivalence** présente l'avantage d'être systématique. En particulier, en prenant en compte les **partitions non valides**, nous pourrons découvrir :

- **Des exigences cachées**
- **D'éventuels défauts**

Mais le tableau 1 met aussi en évidence certaines **faiblesses** de cette technique :

- Il est **peu probable** qu'un enfant de **10 ans** puisse acheter du vin. Les erreurs sont **plus probables aux limites.** Il aurait été plus pertinent de tester un cas où l'**anniversaire des 16 ans est demain.**
- Nous n'avons jamais testé les alcopops. Ces boissons, bien que **faiblement alcoolisées,** sont considérées comme aussi nocives que les liqueurs en raison de leur goût sucré.
- Certaines **combinaisons d'âge et de type de boisson** restent encore non testées.

Dans les sections suivantes, nous verrons comment **d'autres techniques de test permettent d'améliorer la couverture des tests.**

Analyse des valeurs limites - Tester les limites

Les erreurs ont tendance à se produire près des limites d'une plage de valeurs. L'une des raisons en est que les limites ne sont pas toujours définies avec précision. Si nous disons "16 à 17 ans", cela inclut-il 17 ans ou non ?
La réponse est évidente, car la tranche d'âge suivante est "18 ans et plus", mais sans cette information tirée du contexte, nous n'en sommes pas sûrs.

L'analyse des valeurs limites s'appuie sur le partitionnement par équivalence et l'idée est de couvrir toutes les valeurs limites de toutes les partitions d'équivalence.
En supposant que la tranche d'âge de notre exemple soit techniquement fixée à [0..130], nous obtenons les valeurs limites suivantes pour l'âge :

- **Moins de 16 ans**
  - 16ème anniversaire demain
- **16 à 17 ans**
  - 16ème anniversaire aujourd'hui
  - 18ème anniversaire demain
- **18 ans ou plus**
  - 18ème anniversaire aujourd'hui
  - 131ème anniversaire demain
- **Valeurs invalides**
  - pas encore né
  - 131ème anniversaire aujourd'hui

Bonne pratique
Combinez les valeurs des partitions d'équivalence de manière intelligente afin de limiter le nombre de tests. Le plus souvent, il n'est pas nécessaire de tester toutes les combinaisons, mais toutes les valeurs limites doivent être testées au moins une fois.

Comme vous pouvez le constater, nous disposons de cinq cas de test dans des tranches d'âge valables. Cela nous permet de tester les cinq boissons, y compris les alcopops, à condition d'avoir une vue d'ensemble de ce qui reste à tester et de ce qui est déjà couvert.

Arbre de classification - tester toute les feuilles

En particulier si vous avez de nombreux paramètres, il **peut être difficile d'oublier les combinaisons.** C'est pourquoi il est utile de visualiser les options. **La méthode de l'arbre** de classification (appelée technique de l'arbre de classification par l'ISTQB®) est un moyen d'y parvenir. Il **ne s'agit pas d'une technique autonome.** Elle s'appuie sur les deux techniques décrites précédemment et les complète.

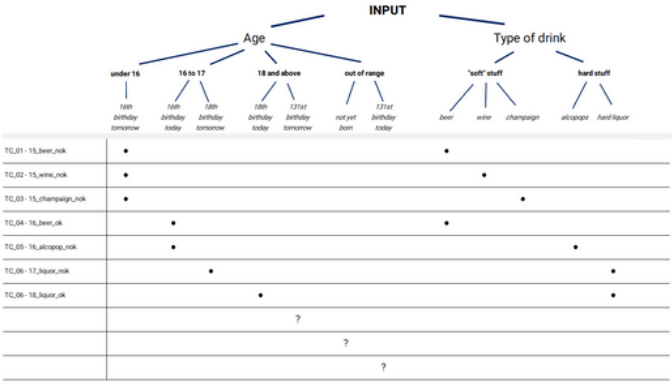


Figure 1 : Arbre de classification (exemple)

La figure 1 montre l'arbre de classification pour notre exemple. Chaque ligne de la partie inférieure représente un cas de test. Chaque puce indique la valeur sélectionnée pour ce paramètre spécifique. La visualisation montre clairement ce qui doit être testé de manière intensive et ce qui est négligé. **Avec l'aide d'outils appropriés, la conception des tests devient facile et l'outil indiquera les valeurs non couvertes,** comme les points d'interrogation dans notre exemple (fictif).
Nous avons délibérément limité l'arbre de classification aux paramètres d'entrée, car il est assez facile de savoir quel sera le résultat attendu. L'ajout du paramètre de sortie à l'arbre ne ferait que le rendre plus confus.

Bonne pratique
Utilisez chaque technique dans la mesure où elle a du sens dans votre contexte

Tables de décision – Test des règles métier

Parfois, nous devons **tester des règles de gestion complexes comportant de nombreuses dépendances.** Dans ce cas, les tables de décision sont la meilleure technique à utiliser.
Les tables de décision contiennent des lignes pour les conditions et les actions. Dans les lignes de conditions, nous définissons des combinaisons de paramètres d'entrée, tandis que les lignes d'actions correspondent au résultat attendu. Chaque colonne de la partie droite du tableau correspond à un cas de test (voir figure 2).

			Rules / Test Cases						
Conditions	Age	under 16	yes	yes	no	no	no	no	no
		16 to 17	no	no	yes	yes	no	no	no
		18 or above	no	no	no	no	yes	yes	no
		not yet born	no	no	no	no	no	no	yes
	Drink	"soft" stuff	yes	no	yes	no	yes	no	no
		hard stuff	no	yes	no	yes	no	yes	yes
Actions	Result	allowed			X		X	X	
		rejected	X	X		X			
	Msg	Error							X

Figure 2 : Conception des tests à l'aide d'une table de décision (exemple)

La figure 2 illustre une façon courante de représenter les tables de décision, mais ce n'est pas la seule. Comme souvent, la manière exacte de mettre en œuvre une certaine technique dépend de l'outil utilisé. Par exemple, la figure 3 montre **une table de décision dans Yest,** l'outil de conception visuelle de tests de Smartesting. Pour une meilleure lisibilité, les lignes et les colonnes sont interverties, c'est-à-dire que chaque ligne correspond à un cas de test. Les colonnes contiennent des conditions et des actions. En outre, il est **possible de spécifier des étapes de test dans Yest qui sont utilisées pour la génération de cas de test.**

	Actions	Expected results
1	Sign in with user name and password	You are redirected to the online shop's main page.
2	As a user < ***16***, try to buy a ***hard stuff***	You are ***rejected***
3	Pay with credit card	Purchase process is successfully completed

Figure 3 : Test de la table de décision avec Yest

2.3 Approches Agile des tests

Développement guidé par le comportement - plus que l'écriture de scripts Gherkin.

Le développement guidé par le comportement (BDD) est devenu très populaire, mais il est souvent mal compris en tant que technique de test. En fait, le BDD est **généralement assimilé à l'écriture de scénarios de test en syntaxe Gherkin.** Mais c'est aussi peu pertinent que de dire qu'une réunion quotidienne constitue un processus Scrum. Pour Gáspár Nagy, auteur de plusieurs ouvrages sur le sujet, "le BDD consiste à comprendre, documenter et vérifier les exigences métier à l'aide de scénarios illustratifs". [source]

Figure 4 : Les trois étapes d'une approche BDD

L'idée sous-jacente est que nous comprenons mieux les choses si elles sont concrètes et pratiques. Les épopées (“Epics”), les récits d'utilisateurs (“User Story”) ou les exigences sont théoriques. Pour les comprendre, nous devons connaître le contexte, à la fois du système testé et des utilisateurs. Nous devons nous parler et nous renseigner sur les détails. La façon la plus simple de le faire est de trouver des exemples, c'est pourquoi BDD est également appelé "spécification par l'exemple".

Bonne pratique
Utiliser la technique de la "cartographie des exemples" pour structurer les conversations pendant la phase de "découverte", pour trouver des exemples et identifier les règles commerciales sous-jacentes. Pour en savoir plus sur cette technique, veuillez consulter le blog original de Matt Wynne.

Ensuite, nous devons documenter ces connaissances. Dans BDD, cette étape est appelée "**Formulation**" (voir figure 4). BDD étant une approche pilotée par les tests, nous utilisons des scénarios de test pour spécifier le comportement attendu. De cette manière, nous faisons d'une pierre deux coups. D'une part, nous documentons le comportement attendu, fournissant ainsi les informations nécessaires aux différentes parties prenantes. D'autre part, nous spécifions nos tests d'acceptation sans effort supplémentaire.

Scenario: Under 16 – beer – rejected
Given a customer enters the store
And the customer is under 16
When the customer wants to by a beer
Then the purchase attempt is rejected

Figure 5 : Les moins de 16 ans

La figure 5 illustre un scénario BDD pour notre magasin d'alcools, rédigé en Gherkin.
Cette syntaxe facilite l'automatisation, chaque ligne correspondant à un mot-clé interprétable par le cadre de test.
Bien menée, cette approche favorise la collaboration entre les "3 Amigos" (Métier, Développement, Test). Cependant, des erreurs courantes peuvent en réduire l'efficacité. Omettre la phase de découverte limite la communication et la collaboration. De plus, des scénarios trop longs deviennent complexes et difficiles à maintenir, surtout pour les tests systèmes ou end-to-end.

Bonne pratique
Envisager les tests basés sur des modèles (Model-Based Testing) pour vérifier les user stories dans leur contexte. Combiner les deux approches dans le cadre du développement guidé par le comportement (Behavior-Driven Development - BDD) pour profiter de l'automatisation à base des mots-clés BDD aux niveaux de test plus élevés.

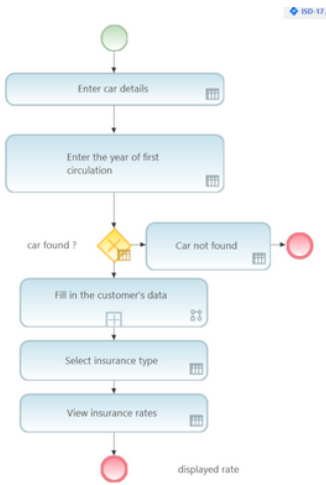
Tout d'abord, il n'existe pas UNE seule méthode de test basé sur un modèle (MBT). Il s'agit en réalité d'un ensemble d'approches. L'ISTQB le définit simplement comme : **"Test basé sur ou impliquant des modèles".**

Cette définition englobe toutes les approches utilisant des modèles à des fins de test, y compris les arbres de classification, mais aussi des modèles textuels écrits dans des langages spécifiques à un domaine.
La grande majorité des approches MBT utilisent toutefois des modèles graphiques.

Cette représentation visuelle offre plusieurs avantages majeurs :

- **Une lecture facilitée :** Les images sont plus faciles à interpréter que le texte et permettent d'avoir un aperçu rapide des flux et des dépendances. Elles constituent une source d'information précieuse pour vérifier certains aspects ou expliquer le contexte à de nouveaux membres de l'équipe.
- **Une accessibilité accrue :** Si la syntaxe du modèle est suffisamment simple, il devient compréhensible même pour des non-experts. Cela brise les silos et rapproche les équipes métiers des équipes techniques. En ce sens, les modèles favorisent la communication et la collaboration.
- **Une approche structurée :** Penser en termes de modèles encourage l'abstraction et renforce la conception des tests. Cela permet de ne pas se focaliser immédiatement sur une description détaillée et linéaire des étapes de test. À la place, on met l'accent sur les enjeux métier, les partitions d'équivalence et les valeurs limites.

Ainsi, le test basé sur modèle (MBT) s'inscrit dans **une démarche plus globale,** visant à **améliorer la qualité et l'efficacité des tests** grâce à une structuration rigoureuse et visuelle.



Bonne pratique
Lorsque vous devez expliquer quelque chose plusieurs fois, dessinez un modèle et joignez-le au ticket auquel il se rapporte.

La figure 6 illustre un **modèle de test dans Yest for Jira,** l'addon gratuit de Smartesting. Yest for Jira permet de lier les workflows Yest aux Epics, fonctionnalités ou autres éléments, ainsi qu'aux tickets Jira (user stories, défauts).

La visualisation est enrichie par des **informations supplémentaires :**

- **Étapes de test et résultats attendus**
- **Mots-clés d'automatisation**
- **Exigences connexes**
- **Niveaux de risque**

L'outil **MBT** exploite les **chemins du modèle**, collecte ces informations et les assemble pour créer des **cas de test détaillés.** Selon les données, ces tests peuvent être exécutés **manuellement et/ou automatiquement**, avec **traçabilité aux exigences et couverture optimisée.**

	Actions	Expected results
1	As a new user, register to online shop	You have to enter your name and birthday.
2	Sign in with user name and password	You are redirected to the online shop's main page.
3	As a user < *16*, try to buy a *"soft" stuff*	You are *rejected*
4	Pay with Paypal	Purchase process is successfully completed

Figure 7: Scénario de tests

La figure 7 montre un scénario de test abstrait obtenu à l'aide de **Yest, l'outil de conception visuelle de tests de Smartesting.** Notez qu'il est également possible de générer des cas de test concrets comprenant des valeurs de données et même des scripts de mots-clés pour l'automatisation des tests à partir du même modèle en utilisant Yest.

Heureusement, il est **possible de construire le modèle MBT** en suivant une approche descendante. Tout d'abord, le modèle général est clarifié et discuté.
Cela permet de préparer le terrain pour l'ensemble de l'équipe du projet. Ensuite, les détails sont ajoutés progressivement, au rythme d'une mise en œuvre agile et itérative.

Bonne pratique
Combinez les avantages du développement piloté par les tests et du MBT en mettant en œuvre Visual ATDD. **Regardez cette vidéo** pour en savoir plus.

Le choix de l'**outil a une influence majeure** sur la nature exacte de l'approche MBT.
Il est donc important d'examiner attentivement ces outils.
Voici quelques aspects à prendre en compte :

Modélisation
- Quelle est la complexité de la syntaxe du modèle ?
- Est-elle adaptée à votre domaine technique et aux parties prenantes impliquées ?
- Est-il possible d'établir une traçabilité entre les exigences et les éléments du modèle ?
- Existe-t-il un support pour lutter contre le syndrome de la "feuille blanche" ?

Collaboration
- Est-il possible de partager les modèles avec d'autres (idéalement sans coûts supplémentaires) ?
- Plusieurs membres de l'équipe peuvent-ils contribuer au même modèle ?
- Quelle est la qualité de la gestion de la configuration ?

Génération de tests
- Les tests générés sont-ils adaptés à vos besoins ?
- L'outil MBT s'interface-t-il avec vos outils de gestion des tests et d'automatisation des tests ?
- Les résultats de l'exécution des tests sont-ils rapportés dans le modèle ?

Maintenance
- Les tests existants sont-ils remplacés en cas de modification du modèle ? Est-ce acceptable dans votre contexte ?
- L'outil fournit-il une aide supplémentaire pour la maintenance du modèle et des tests ?

Bonne pratique
Collaborez autour de représentations visuelles pour optimiser votre activité de conception de tests avec **Yest by Smartesting**. Accélérez de 40% la conception des tests fonctionnels les plus pertinents et leur mise en œuvre pour une exécution manuelle et automatisée.

2.4 Pourquoi adopter le Model Based Testing avec Yest ?

L'utilisation de **Yest** pour le **Model-Based Testing** (MBT) optimise le processus de test à plusieurs niveaux :

- **Visualisation claire** des scénarios de test pour une meilleure compréhension des exigences et une communication facilitée au sein de l'équipe projet.
- **Génération automatique** de tests pertinents basés sur les règles métier, assurant une couverture optimale grâce à leur traçabilité dans les modèles.

Yest accélère aussi considérablement la conception et la maintenance des tests grâce à des fonctionnalités avancées : mise à jour en masse, autocomplétion, recherche intelligente, analyse d'impacts, propagation contrôlée des changements... **Nos clients constatent une réduction de 40 % du temps consacré à ces phases.**

En plus de cette accélération, **Yest simplifie l'automatisation.** Grâce au **Keyword-Driven Testing,** la transition des tests manuels vers l'automatisation devient intuitive.

L'approche par modèles peut dérouter certains testeurs au début. C'est pourquoi nous mettons à disposition des **ressources en ligne et un accompagnement personnalisé** via nos Customer Success Managers, pour vous guider vers une maîtrise experte et efficace de Yest.

2.5 Comment adopter le Model Based Testing avec Yest ?

Adopter le MBT avec Yest est relativement simple. Il peut se faire simplement et rapidement en respectant les contraintes projets.

Voici les étapes à suivre pour une **montée en compétence efficace sur Yest :**

- **Étape 1 :** Installation de Yest
- **Étape 2 :** L'initiation à Yest
- **Étape 3 :** Génération de tests pour votre projet

Étape 1 : Installation de Yest

L'installation est **simple et rapide.** Inscrivez-vous sur la **communauté Yest** pour accéder au produit, à sa documentation et à une licence d'essai. Plus vous pratiquez, plus vite, vous deviendrez expert !

Étape 2 : Découverte de Yest

Prenez entre **1 heure et une demi-journée** pour vous familiariser avec l'outil :

- **En 1h,** générez vos premiers tests à partir d'un modèle.
- **En une demi-journée,** explorez les différentes fonctionnalités pour concevoir des tests exécutables avec des jeux de données.

Adaptez votre apprentissage selon votre **temps disponible et vos besoins.**

Étape 3 : Génération de tests pour votre projet

Dès que vous comprenez le mécanisme de génération et publication des tests, vous pouvez concevoir vos premiers parcours et tests avec Yest.

- **Commencez petit :** ciblez une **spécification ou une User Story (US)** et créez votre premier modèle.
- **Montez en puissance :** à chaque itération, augmentez le volume de spécifications modélisées et u**tilisez progressivement Yest à 100 %** pour la conception des tests.

Cette approche progressive **limite les risques de retard et optimise votre processus** de test. Après quelques itérations, vous constaterez un **gain de temps** significatif, notamment sur la mise à jour des tests.

Découvrir Yest

III. Cas d'usage

3.1 Yest dans l'IT

Dans l'IT, chaque application a ses spécificités, mais les tests suivent souvent un schéma commun. En général, ces applications (web ou desktop) permettent à l'utilisateur de saisir des informations (formulaires, commandes, devis...) et génèrent en retour une **synthèse et un enregistrement des données.**

Tester une application consiste donc à **simuler le comportement d'un utilisateur** final en décrivant :

- **Les actions réalisées** sur l'interface.
- **Les résultats attendus pour** vérifier la conformité aux spécifications et aux besoins utilisateurs.

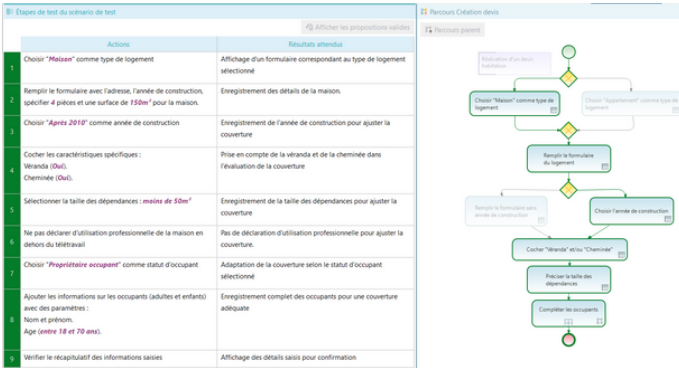
Le principal défi ? La complexité des combinaisons à tester.

- Différents profils d'utilisateurs avec des droits d'accès variables.
- Multiples combinaisons de données influençant l'affichage et le comportement de l'application.

Il est essentiel de **sélectionner des combinaisons pertinentes** et d'accéder aux données nécessaires. La clé réside dans une **conception rigoureuse des tests,** bien distincte de leur implémentation, afin d'optimiser leur efficacité et d'éviter les redondances.

Avec Yest, la représentation des tests sous forme de **parcours applicatifs** permet de mieux refléter l'usage des utilisateurs.

1. **Description du scénario :** le parcours illustre les actions de l'utilisateur.
2. **Complétion des blocs :** des tables de décision identifient les combinatoires à tester.
3. **Définition des cas métiers :** les données de conception sont classées en équivalences (ex. : prêt<1000 €, entre 1000 € et 3500 €, etc.), permettant de tester des comportements variés.
4. **Génération des jeux de données :** chaque classe est associée à des valeurs concrètes (ex. : 999 €, 1200 €…).



Étapes de test du scénario de test

Afficher les propositions valides

	Actions	Résultats attendus
1	Choisir "**Maison**" comme type de logement	Affichage d'un formulaire correspondant au type de logement sélectionné
2	Remplir le formulaire avec l'adresse, l'année de construction, spécifier 4 pièces et une surface de 150m² pour la maison.	Enregistrement des détails de la maison.
3	Choisir "**Après 2010**" comme année de construction	Enregistrement de l'année de construction pour ajuster la couverture
4	Cocher les caractéristiques spécifiques : Véranda (**Oui**). Cheminée (**Oui**).	Prise en compte de la véranda et de la cheminée dans l'évaluation de la couverture
5	Sélectionner la taille des dépendances : **moins de 50m²**	Enregistrement de la taille des dépendances pour ajuster la couverture
6	Ne pas déclarer d'utilisation professionnelle de la maison en dehors du télétravail	Pas de déclaration d'utilisation professionnelle pour ajuster la couverture.
7	Choisir "**Propriétaire occupant**" comme statut d'occupant	Adaptation de la couverture selon le statut d'occupant sélectionné
8	Ajouter les informations sur les occupants (adultes et enfants) avec des paramètres : Nom et prénom. Age (**entre 18 et 70 ans**).	Enregistrement complet des occupants pour une couverture adéquate
9	Vérifier le récapitulatif des informations saisies	Affichage des détails saisis pour confirmation

Parcours Création devis

Parcours parent

Illustration de la conception des tests avec Yest sur un exemple IT

Au-delà de la production de tests, les **modèles Yest** sont une documentation vivante partageable avec les parties prenantes. Chaque étape (modélisation, conception des tables et jeux de données, génération des tests) peut être revue et validée, garantissant une **compréhension alignée des besoins métier** dès le début du projet.

3.2 Yest dans l'industrie

L'**industrie** englobe les **activités produit** des entreprises, distinctes des activités IT classiques (gestion, RH…). Bien que les contextes varient, **certaines exigences restent constantes :**

- **Traçabilité stricte des exigences,** parfois imposée par des normes.
- **Automatisation élevée** des tests par rapport à l'IT.
- **Environnements de test diversifiés** (SIL, HIL, simulations…).
- **Utilisation fréquente de modèles** en ingénierie système.

Avec ce niveau d'automatisation, **donner de la visibilité sur les tests devient essentiel** pour les équipes **Vérification & Validation et Développement.**

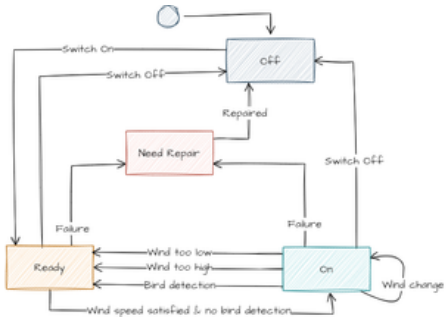
👉 **Yest et Yest for Jira répondent à ces enjeux :**

- **Vue globale** pour tous grâce aux parcours applicatifs dans Yest for Jira.
- **Détail et automatisation** gérés dans Yest.
- **Traçabilité complète** de la prise en compte des exigences jusqu'à la publication des tests.
- **Distinction claire entre conception et implémentation,** permettant un démarrage anticipé (shift-left) et une adaptation aux différents environnements d'exécution.

Tests sur machines à états dans l'industrie
Les systèmes industriels utilisent souvent des **machines à états** pour formaliser leur comportement.

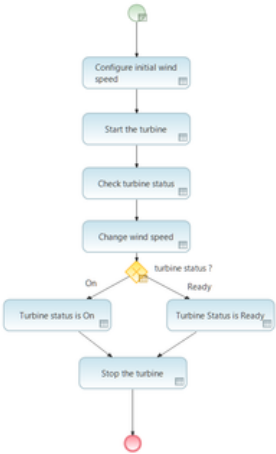
Yest offre deux approches pour les tester :

1. **Reproduction exacte** de la machine à états en parcours Yest → couverture complète des transitions, mais faible lisibilité.
2. **Parcours orienté test** → meilleure compréhension et contrôle des objectifs de test. Cette approche permet aussi de **revoir la spécification elle-même**, en évaluant son usage sous un angle plus fonctionnel.



Représentation transcrite dans Yest (les états et les transitions sont toutes représentées, les transitions sont figurées sous forme de tâches à effectuer au moment du test)

Représentation plus orientée suite d'actions de test se focalisant sur une partie de la spécification (comportement en fonction de la vitesse du vent)



Yest for Jira, une révolution pour l'ingénierie des tests

Yest for Jira est une application gratuite qui transforme la gestion des exigences et des tests dans Jira.

💡 **Pourquoi l'adopter ?**

✓ **Éditeur de workflow intuitif :** créez des logigrammes facilement.
✓ **Connexion aux éléments Jira :** associez actions, User Stories, Defects, Epics et Features.
✓ **Synchronisation bidirectionnelle avec Yest Enterprise :**

- Les workflows des business analysts et product owners servent à générer des cas de test.
- Toute l'équipe accède aux **critères d'acceptation** définis dans Yest Enterprise.

Compatibilité avec Xray et Zephyr : suivez en temps réel la progression des tests dans vos workflows.

Avec Yest for Jira, l'ingénierie des tests devient plus claire, rapide et efficace. 🚀

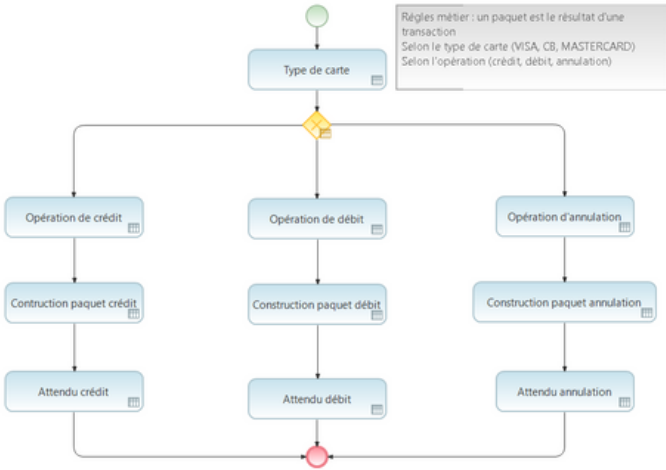
3.3 Yest batch, API

Les **tests de batch** consistent à injecter des **blocs de données** dans des flux et à vérifier la pertinence des résultats obtenus par les récepteurs.

Objectif : tester **différentes configurations** de blocs d'entrée pour couvrir un maximum de cas et valider les résultats attendus.

- **Défi principal :** la constitution des blocs de données, où réside la complexité fonctionnelle.
- **Solution avec Yest :** une **construction maîtrisée des combinatoires** pour **optimiser la couverture** tout en réduisant le nombre de tests.

Avec **Yest,** la représentation sous forme de **parcours applicatif** ne simule pas les actions utilisateur, mais l'assemblage des blocs de données. Ce parcours peut se limiter à leur construction et à la définition des résultats attendus, ou aller jusqu'à la **soumission des flux.**



3.4 Utilisation de Yest pour le test d'API

On se contente souvent de **tests unitaires** pour les API, mais des **tests plus complets** apportent plusieurs bénéfices :

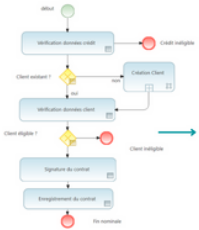
1. **Réduction des tests IHM :** tester directement la couche API permet d'éviter l'incertitude liée aux tests automatisés sur l'interface graphique (risque de faux positifs/négatifs).
2. **Accès à des contextes inatteignables en unitaire :** certains enchaînements API permettent de tester des situations impossibles à simuler autrement.

💡 **Yest facilite les tests d'API sans expertise technique :**

- **Parcours et scénarios lisibles** et maintenables par des testeurs fonctionnels.
- **Implémentation prise en charge** par un automaticien si nécessaire.
- **Création et gestion simplifiée** des contextes applicatifs nécessaires aux appels API.
- **Optimisation des combinatoires** pour maximiser la couverture.

Les tests peuvent être exécutés via des **outils spécialisés** (Postman, SOAP UI) ou des frameworks généralistes (Robot Framework, JUnit), **tous compatibles avec Yest** à partir d'une même description de parcours et de scénarios.

Exemple : d'un modèle à des tests API



	Actions	Résultats attendus
1	Vérification données crédit : Montant : 780 Durée : *Entre 6 et 24 mois* Type de bien : *Son, Vidéo*	Vérifier que le crédit est : Validé
2	Vérification données client : Age : *De 14 à 60 ans* Profession : Salarié	Vérifier que le client est : Validé
3	Signature du contrat	
4	Enregistrement du contrat	Le contrat doit être correctement enregistré

```
POST {{creditserver}}/resetcustomers
POST {{creditserver}}/resetcontracts
GET {{creditserver}}/checkcredit
GET {{creditserver}}/checkcustomer
POST {{creditserver}}/contract
POST {{creditserver}}/sign/v1
POST {{creditserver}}/finalize
POST {{creditserver}}/resetcustomers
POST {{creditserver}}/resetcontracts
```

Conclusion

Nous espérons que cet ebook vous a apporté une **compréhension solide du test design** et de son rôle dans le développement logiciel.

Ce que nous avons couvert :

- Les concepts fondamentaux du test design.
- Les techniques de test les plus efficaces.
- Les avantages du Model-Based Testing avec Yest.

🚀 **Pourquoi adopter Yest ?**

- Amélioration de la qualité des tests.
- Réduction des erreurs.
- Accélération du cycle de développement.

Le **test design est en constante évolution,** et nous vous encourageons à **explorer de nouvelles** approches et à vous tenir informé des dernières tendances du secteur.

Merci de votre lecture et bon succès dans vos projets avec Yest !
Si vous avez des questions ou des commentaires, contactez-nous.

Smartesting®

Testez ce qui compte vraiment

Solutions de Test **Intelligentes**

Yest® Test Design collaboratif

Gravity Insights d'utilisation

Formation : IA Générative pour les tests

www.smartesting.com