Le carré des contraintes

Au-delà du triangle d'or : le modèle à quatre sommets de Jurgen Appelo pour piloter un projet.

5TTR 6TTR 6GMS 4TTR

Pourquoi ce cours ?

Dans n'importe quel projet en équipe, tu vas vivre la même tension : trop à faire, pas assez de temps, une équipe imparfaite, et un client (ou un prof, ou un public) qui veut quand même un résultat de qualité.

Ce cours te donne un **outil mental** pour rendre ces tensions visibles, en parler, et faire des choix lucides. Cet outil s'appelle le **carré des contraintes**, proposé par Jurgen Appelo dans *Management 3.0* (2010) en évolution du classique **triangle d'or**.

Principe fondamental

> SI UN SOMMET VARIE DANS UN SENS, SOIT LES 2 SOMMETS ADJACENTS VARIENT DANS LE MÊME SENS, SOIT LE SOMMET OPPOSÉ VARIE DANS LE SENS INVERSE.

C'est la règle unique qui régit toute la mécanique du modèle. Le reste du cours en dérive.

Objectifs d'apprentissage

À la fin de cette séquence, tu seras capable de :

1. **Identifier** les quatre sommets du carré des contraintes et donner leur équivalent en anglais et en français.
2. **Expliquer** le principe fondamental d'arbitrage : adjacents même sens / opposé sens inverse.
3. **Décomposer** la notion de ressources entre les outils et la capacité humaine (formule à quatre facteurs).
4. **Analyser** un projet en identifiant le sommet modifié et les sommets compensateurs.
5. **Argumenter** un choix d'arbitrage dans une réunion d'équipe agile en s'appuyant sur le carré.

Cinq notions-clés à retenir

#	Notion	Idée centrale
1	**Triangle d'or**	Le modèle classique : Scope, Time, Cost. Critiqué par Appelo car la qualité y est implicite.
2	**Carré des contraintes**	Quatre sommets explicites : **Scope, Time, Quality, Resources**.
3	**Adjacents vs opposé**	Si je modifie un sommet, soit les deux adjacents bougent dans le même sens, soit l'opposé bouge en sens inverse.
4	**Ressources = outils × humains**	Une équipe sans outils ne produit pas, des outils sans équipe non plus. La capacité humaine, elle, suit la formule `N × C`
5	**Pas d'arbitrage gratuit**	Aucun projet ne maximise les quatre sommets en même temps. Le rôle du chef de projet est de rendre les arbitrages **ex**

1. Du triangle au carré

Le triangle d'or classique (Iron Triangle)

Depuis les années 1960, la gestion de projet se résume souvent à trois contraintes :

- **Scope** (périmètre) — ce qu'on doit livrer
- **Time** (délai) — combien de temps on a
- **Cost** (coût) — combien ça coûte

> *« Vite, pas cher, beaucoup de fonctionnalités — choisis-en deux. »*

C'est élégant, mais Appelo trouve que **la qualité y est traitée comme un sous-produit silencieux**. Elle se dégrade quand on serre les trois autres, mais personne n'en parle ouvertement.

Le carré d'Appelo

Appelo propose d'ajouter explicitement la **Qualité** comme quatrième sommet :

Voisinages :

Sommet	Adjacents (côté du carré)	Opposé (diagonale)
Scope	Time, Resources	Quality
Time	Scope, Quality	Resources
Quality	Time, Resources	Scope
Resources	Scope, Quality	Time

Cette topologie est essentielle : c'est elle qui détermine *qui* doit bouger *quand* tu modifies un sommet.

2. Les quatre sommets en détail

↖ Sommet haut-gauche : Scope *(Périmètre)*

Définition. L'ensemble des fonctionnalités, livrables et exigences qui définissent **ce que** le projet doit produire. C'est la *frontière* du projet : ce qui est inclus, ce qui est exclu.

Vocabulaire.

- GB *Scope, requirements, features, deliverables*
- FR Périmètre, exigences, fonctionnalités, livrables

Exemple. Un site de réservation avec catalogue, calendrier, paiement en ligne, comptes utilisateurs, scan QR et back-office → **scope large**. Le même site avec uniquement le catalogue et le calendrier → **scope réduit**.

Piège classique. Le **scope creep** : le périmètre grossit en douce au fil du projet (« on pourrait juste ajouter… »). Sans arbitrage, c'est le délai, la qualité ou les ressources qui paient l'addition.

↗ Sommet haut-droite : Time *(Délais)*

Définition. La durée totale disponible pour livrer le projet. Inclut la **date de fin** (deadline), les **jalons** intermédiaires, et le **rythme** de travail.

Vocabulaire.

- GB *Time, schedule, deadline, milestones*
- FR Délais, temps imparti, échéance, jalons

Exemple. Une soirée d'entreprise est prévue le 14 février. Cette date **ne bouge pas**. Si le client demande une nouvelle catégorie de questions trois semaines avant, c'est le scope ou la qualité qui devra s'ajuster — jamais le délai.

Particularité. Le temps est souvent **la contrainte la plus rigide** : un salon, une rentrée scolaire, une obligation légale, un lancement marketing — ces dates sont externes au projet et ne se négocient pas.

↘ Sommet bas-droite : Quality *(Qualité)*

Définition. Le niveau d'exigence du livrable : robustesse du code, ergonomie, accessibilité, performance, sécurité, design, absence de bugs, conformité aux normes.

Vocabulaire.

- GB *Quality, robustness, usability, performance*
- FR Qualité, robustesse, ergonomie, performance

Exemple. Une app médicale doit être validée **sans bug critique** — la qualité est imposée par la loi. Un prototype présenté en interne peut tolérer des bugs mineurs : on ajuste après.

Le piège de la qualité sacrifiée. Quand on dit *« on livrera vite, on corrigera après »*, on crée de la **dette technique** : du code mal écrit qu'il faudra payer (avec intérêts) plus tard. Appelo insiste : rendre la qualité explicite, c'est la rendre négociable **en conscience**, pas par défaut.

↙ Sommet bas-gauche : Resources *(Ressources)*

Définition. Tout ce qui est mobilisé pour réaliser le projet : équipe humaine, outils, matériel, budget, locaux.

Vocabulaire.

- GB *Resources, people, tools, capacity, budget*
- FR Ressources, équipe, outils, capacité, budget

C'est le sommet **le plus mal compris** — et c'est sur lui qu'on va passer un peu plus de temps.

---

3. Zoom — Décomposer les ressources

> ⚠️ **NOTE IMPORTANTE.** Ce qui suit est un **enrichissement personnel** du modèle d'Appelo, **pas la version d'origine**. Dans *Management 3.0*, Appelo traite les ressources de manière assez globale — équipe et budget réunis. La décomposition ci-dessous est une grille de lecture plus fine pour mieux piloter ce qu'il y a derrière la simple étiquette « ressources ».

Deux grandes familles

Pour un projet, les ressources se regroupent en deux grandes familles :

1. **Les outils** (`O`) — tout ce qui n'est pas humain mais qui permet de produire : matériel (ordinateurs, serveurs, hardware spécialisé), logiciels (IDE, suites de design, outils de build), infrastructure (cloud, CI/CD, bases de données), locaux, licences, accès aux specs et à la documentation, budget.
2. **La capacité humaine** (`H`) — ce que l'équipe peut réellement produire, qui dépend de quatre facteurs détaillés ci-dessous.

Les deux dimensions se combinent de manière **multiplicative** :

> $R = O \times H$

Pourquoi multiplicatif ? Parce qu'une équipe top sans serveur, sans IDE, sans accès aux specs → bloquée. Et des outils flambant neufs sans personne pour les utiliser → rien ne sort. Si l'une des deux dimensions tend vers zéro, l'ensemble tend vers zéro.

La capacité humaine : quatre facteurs

> $H = N \times C \times M \times T$

Où :

- **N — Nombre de personnes** dans l'équipe (de 1 à n)
- **C — Compétence** : ce qu'elles savent faire dans le domaine du projet
- **M — Motivation** : leur envie réelle d'avancer sur *ce* projet
- **T — Temps disponible** : la part de leur temps de travail qu'elles peuvent vraiment y consacrer

⚠️ Une remarque essentielle sur les pourcentages

Dans les exemples qui suivent, on utilise des pourcentages (« compétence à 95 % », « motivation à 0 % »…). **Ces chiffres ne sont pas des mesures absolues.** Personne ne calcule scientifiquement « 78 % de motivation » ou « 42 % de compétence » — c'est impossible.

Les pourcentages sont là **uniquement pour faciliter la compréhension** : ils permettent de visualiser intuitivement les ordres de grandeur et l'effet multiplicatif. En pratique, trois niveaux suffisent souvent — *élevé / moyen / faible* — voire juste la question *« est-ce qu'il manque quelque chose ? »*.

L'important n'est **pas la précision du chiffre**. C'est de reconnaître que **tous les facteurs jouent en même temps**, et que **tout facteur très bas fait chuter l'ensemble**.

Pourquoi c'est *multiplicatif* et pas *additif*

C'est la clé du modèle. Une multiplication a une propriété particulière : **si un seul facteur est à zéro, le résultat est zéro**, peu importe les autres.

Regardons quatre cas pour s'en convaincre :

Cas 1 — Le développeur expert mais démotivé

> 1 personne · compétence très élevée · **MOTIVATION NULLE** · temps total disponible → $H \approx 1 \times 0{,}95 \times 0 \times 1 = 0$

Le projet n'avance pas. Aucune productivité. La compétence sans motivation = zéro. C'est **la grande leçon d'Appelo** : la motivation n'est pas un « bonus » qu'on ajoute, c'est un **facteur multiplicatif**. Sans elle, tout le reste est neutralisé.

Cas 2 — L'équipe motivée mais sans compétence

> 5 personnes · **COMPÉTENCE TRÈS FAIBLE** · motivation maximale · temps total disponible → $H \approx 5 \times 0{,}05 \times 1 \times 1 = 0{,}25$

L'enthousiasme ne remplace pas le savoir-faire. Une équipe ultra-motivée mais qui découvre une techno la veille du sprint ne livrera pas une plateforme robuste.

Cas 3 — L'expert sans temps disponible

> 1 personne · compétence maximale · motivation maximale · **TEMPS DISPONIBLE TRÈS FAIBLE** (autres projets prioritaires) → $H \approx 1 \times 1 \times 1 \times 0{,}05 = 0{,}05$

Un excellent profil à quelques % de son temps fournit la même chose qu'un débutant à quelques % : presque rien. Le temps disponible est crucial — et souvent surestimé.

Cas 4 — L'équipe équilibrée

> 5 personnes · compétence assez bonne · motivation bonne · temps partiel → $H \approx 5 \times 0{,}7 \times 0{,}8 \times 0{,}6 \approx 1{,}7$

Soit environ **17 %** d'un maximum théorique de 10 personnes à plein régime sur tous les facteurs. C'est une équipe **correcte et réaliste** : on ne travaille jamais à fond sur tous les axes en même temps.

Comment évaluer ta propre équipe

Avant chaque sprint, pose-toi (en équipe) les quatre questions :

1. **N —** Sommes-nous combien à vraiment travailler dessus ? (Distinguer les présents des actifs.)
2. **C —** Maîtrisons-nous les technos nécessaires, ou faut-il apprendre en chemin ?
3. **M —** Est-ce que tout le monde *a envie* d'avancer ? Y a-t-il un sujet bloquant ?
4. **T —** Combien d'heures réelles peut-on y consacrer cette semaine ?

Et n'oublie pas la première moitié de la formule : **a-t-on les outils nécessaires ?** Sans accès au serveur de test, sans la bonne licence logiciel, sans la doc à jour, même la meilleure équipe sera bloquée.

Si l'un des facteurs (outils inclus) est très bas → c'est lui qu'il faut adresser **en priorité** avant d'ajouter du scope.

---

4. La mécanique d'arbitrage

C'est le **cœur du modèle**. Pour chaque situation, on applique le principe fondamental : **soit** les deux adjacents bougent dans le même sens (🟢 vert), **soit** l'opposé bouge en sens inverse (🔴 rouge).

Situation A — Le périmètre augmente (Scope ↑)

> *Un client ajoute une fonctionnalité en cours de projet.*

🟢 **Soit les deux adjacents augmentent :**

- ↑ **Time** — il faut plus de temps
- ↑ **Resources** — il faut plus de ressources

🔴 **Soit l'opposé diminue :**

- ↓ **Quality** — la qualité baisse en compensation

Situation B — Le délai se raccourcit (Time ↓)

> *La direction avance la deadline de trois semaines.*

🟢 **Soit les deux adjacents diminuent :**

- ↓ **Scope** — on réduit les fonctionnalités
- ↓ **Quality** — on accepte une qualité moindre

🔴 **Soit l'opposé augmente :**

- ↑ **Resources** — on ajoute des renforts

Situation C — Les ressources diminuent (Resources ↓)

> *Un développeur quitte l'équipe à mi-projet.*

🟢 **Soit les deux adjacents diminuent :**

- ↓ **Scope** — on coupe des features
- ↓ **Quality** — la qualité chute

🔴 **Soit l'opposé augmente :**

- ↑ **Time** — on allonge le délai

Situation D — La qualité doit monter (Quality ↑)

> *Audit qualité imposé, ou cible plus exigeante.*

🟢 **Soit les deux adjacents augmentent :**

- ↑ **Time** — il faut plus de temps pour peaufiner
- ↑ **Resources** — plus de relecture, plus de tests

🔴 **Soit l'opposé diminue :**

- ↓ **Scope** — on livre moins, mais mieux

Le rôle du chef d'équipe

À chaque demande de changement, **trois questions à se poser à voix haute** :

1. Quel sommet veut bouger, et dans quel sens ?
2. Qui sont les adjacents, qui est l'opposé ?
3. Quel scénario de compensation choisit-on : **adjacents même sens** ou **opposé sens inverse** ?

L'erreur classique : *« on garde tout pareil et on encaisse »*. Ça n'existe pas. Le carré rend l'arbitrage **explicite** au lieu de le subir en silence.

---

5. Lien avec l'écoresponsabilité numérique

Un sommet souvent oublié mais qui devrait être discuté : **toute fonctionnalité ajoutée a un coût environnemental**.

- Plus de scope → plus de code, plus de serveurs, plus de stockage, plus de bande passante consommée.
- Plus de ressources humaines → plus de déplacements, de hardware, de chauffage de bureaux.
- Plus de qualité (notamment perf) → souvent moins de gaspillage énergétique côté utilisateur.

Quand tu négocies le scope en équipe, pose la question : **cette fonctionnalité justifie-t-elle son empreinte ?** Le carré devient alors aussi un outil d'arbitrage éthique. C'est la posture **Lean & Green** de la gestion de projet moderne.

---

6. Exercices

Exercice 1 — Application du principe

Pour chaque scénario, indique les **deux scénarios de compensation possibles** selon le principe fondamental :

1. La qualité doit augmenter (audit imposé).
2. Le scope doit diminuer (le client coupe une feature).
3. Les ressources doivent augmenter (recrutement d'un dev).
4. Le délai s'allonge (livraison repoussée).

Réponse attendue par scénario :

- 🟢 **Adjacents :** quels sommets et dans quel sens ?
- 🔴 **Opposé :** quel sommet et dans quel sens ?

Exercice 2 — Diagnostic d'équipe

Évalue la capacité réelle de ton équipe agile pour la semaine prochaine. Pour chaque facteur, contente-toi d'un niveau *élevé / moyen / faible* (pas besoin de pourcentages précis) :

- **Outils** disponibles (matériel, logiciels, accès aux specs)
- **N** — nombre de personnes vraiment actives
- **C** — compétence collective sur les technos requises
- **M** — motivation
- **T** — temps réellement disponible

→ Identifie le **maillon faible** (le facteur le plus bas) et propose **une action concrète** pour le faire remonter.

Exercice 3 — Négociation

Le client demande une fonctionnalité supplémentaire à 10 jours de la livraison. Rédige un message Slack ou e-mail expliquant **les deux options possibles** (adjacents même sens / opposé sens inverse) et **ta recommandation argumentée**.

Exercice 4 — Étude de cas

Choisis un projet réel que tu connais (un projet open source, le développement d'un jeu vidéo connu, un grand chantier public…). Analyse-le à travers le carré :

- Quel sommet a été **préservé à tout prix** ?
- Quel sommet a été **sacrifié** ?
- Quel scénario d'arbitrage a été appliqué (adjacents même sens, ou opposé sens inverse) ?

Pour aller plus loin

- **Jurgen Appelo**, *Management 3.0 — Leading Agile Developers, Developing Agile Leaders*, Addison-Wesley, 2010 (chap. 1 & 4).
- **Jurgen Appelo**, *Managing for Happiness*, Wiley, 2016 (sur la motivation comme facteur multiplicatif).
- **The Mythical Man-Month**, Fred Brooks, 1975 (la fameuse loi : *« ajouter des développeurs à un projet en retard le retarde davantage »* — illustration du modèle multiplicatif).
- **The Phoenix Project**, Gene Kim et al. (sur la dette technique et l'arbitrage qualité dans un contexte DevOps).

« Le triangle vous dit ce que vous pouvez optimiser. Le carré vous rappelle ce que vous sacrifiez. »