LOG680
Introduction à l'approche DevOps

Créer des processus de revue et de coordination pour améliorer la qualité de notre travail

The DevOps Handbook

Part IV, Chap 18

Objectifs d'apprentissage

- Expliquer le processus en cinq étapes de GitHub Flow
- Expliquer les dangers associés aux processus d'approbation de changement. Comment peut-on y remédier?
- Expliquer en quoi consiste la révision des modifications par les pairs. Quelles sont ses avantages et inconvénients?
- Expliquer en quoi consiste le principe de programmation en pair (Pair Programming"). Quels sont ses avantages et inconvénients?
- Expliquer en quoi consiste le principe de Pull Request. Quels sont les avantages d'utiliser des Pull Request? Expliquer ce qui constitue une mauvaise et une bonne pull request

Sujets

- Introduction
- GitHub "Pull Request"
- Dangers des processus d'approbation de changement
- Dangers du «contrôle total des changements»
- Coordination et planification des modifications
- Révision par les pairs
- Pair Programming
- Processus de Pull Request
- Élimination des processus bureaucratiques
- Conclusion
- Conclusion : Partie IV

- **Introduction**
- GitHub "Pull Request"
- Dangers des processus d'approbation de changement
- Dangers du «contrôle total des changements»
- Coordination et planification des modifications
- Révision par les pairs
- Pair Programming
- Processus de Pull Request
- Élimination des processus bureaucratiques
- Conclusion
- Conclusion : Partie IV

Introduction

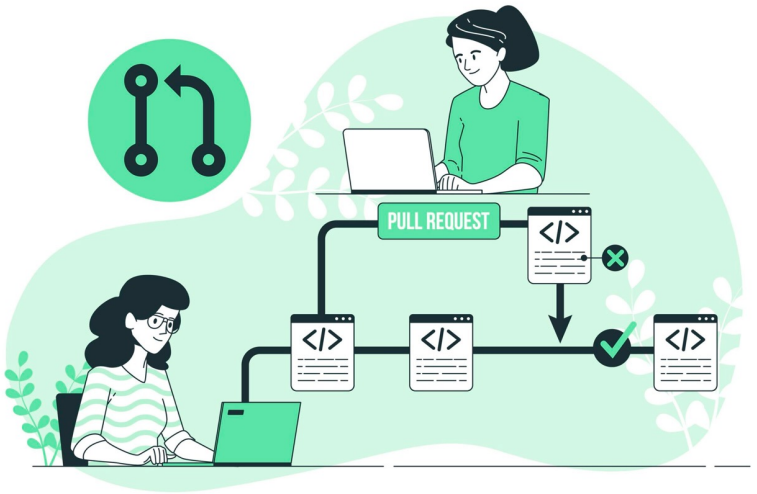
- Dans les chapitres précédents, nous avons
  - **Créé la télémétrie** nécessaire pour voir et résoudre les problèmes liés à la production et **à toutes les étapes de notre pipeline de déploiement**
  - **Créé des boucles de rétroaction rapides des clients** pour nous aider à améliorer l'apprentissage organisationnel
    - Apprentissage qui incite à prendre la responsabilité de la satisfaction du client et de la performance des fonctionnalités, ce qui nous aide à réussir

- Dans ce chapitre**,** notre objectif est de permettre à Dev and Ops de **réduire les risques de modifications de la production avant qu'elles ne soient apportées**
  - Traditionnellement, lorsque nous examinons les modifications pour le déploiement, nous avons tendance à nous fier fortement aux revues, inspections et approbations juste avant le déploiement
  - Ces approbations sont souvent données par des équipes externes qui sont souvent trop éloignées du travail pour pouvoir décider en connaissance de cause si un changement est risqué ou non, et le temps requis pour obtenir toutes les approbations nécessaires rallonge également nos délais de changement

- Introduction
- **GitHub "Pull Request"**
- Dangers des processus d'approbation de changement
- Dangers du «contrôle total des changements»
- Coordination et planification des modifications
- Révision par les pairs
- Pair Programming
- Processus de Pull Request
- Élimination des processus bureaucratiques
- Conclusion
- Conclusion : Partie IV

GitHub "Pull Request"

- GitHub "Pull Request: une des formes les plus populaires de révision par les pairs
  - Exemple de mécanisme de révision par les pairs qui permet d'améliorer la qualité, sécuriser les déploiements et s'intégrer dans le flux de travail quotidien de chacun
  - Peut être utiliser par Dev et Ops



- Scott Chacon (CIO et cofondateur de GitHub)

  « Le concept de "Pull Request" est le mécanisme qui permet aux ingénieurs de **faire part aux autres des modifications** qu'ils ont placées dans un référentiel sur GitHub.

  Une fois qu'une Pull Request est crée, les parties intéressées peuvent **examiner** l'ensemble des modifications, **discuter** des modifications et même **appliquer des validations** de suivi si nécessaire.

  Les ingénieurs qui soumettent une Pull Request demandent souvent un «+1», un «+2», etc., en fonction du nombre d'avis requis, ou d'ingénieurs «@mention» auprès desquels ils aimeraient obtenir des avis. »

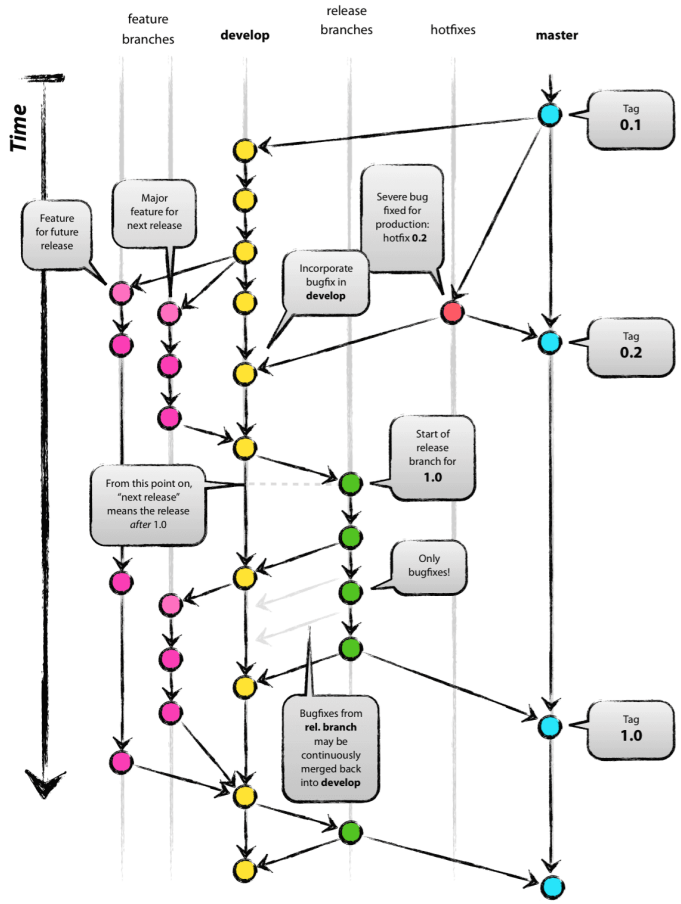
GitHub "Pull Request"

- Les Pull Request sont également le **mécanisme utilisé pour déployer du code en production** grâce à un ensemble de pratiques appelées «GitHub Flow»
  - C'est ainsi que les ingénieurs demandent des revues de code, rassemblent et intègrent des commentaires et annoncent que le code sera déployé en production (c'est-à-dire, branche "principale")
- GitHub Flow est composé de cinq étapes:

1. L'ingénieur crée une branche nommée de manière descriptive à partir du maître ("master"), par exemple, «new-oauth2-scopes»
2. L'ingénieur s'engage localement dans cette branche, en transférant régulièrement son travail vers la même branche nommée sur le serveur
3. Lorsqu'elle a besoin d'informations ou d'aide, ou lorsqu'elle pense que la branche est prête à fusionner, elle ouvre une "pull request"
4. Une fois qu'elle a obtenu les commentaires souhaités et obtenu les approbations nécessaires pour la fonctionnalité, l'ingénieur peut alors la fusionner sur le tronc/maître
5. Une fois les modifications de code fusionnées et poussées vers le maître, l'ingénieur les déploie en production

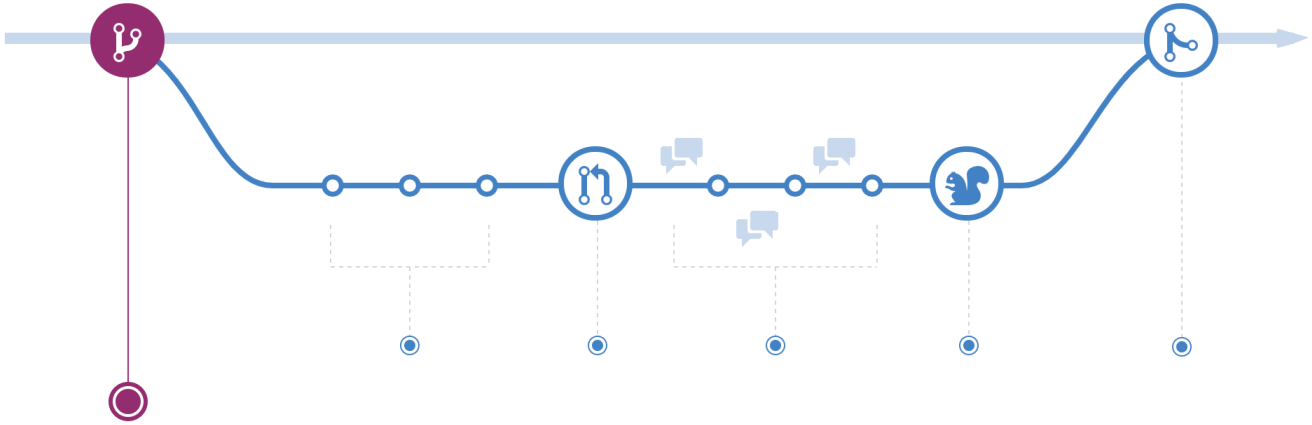
Figure 40: *Comments and suggestions on a GitHub pull request (Source: Scott Chacon, "GitHub Flow," ScottChacon.com, August 31, 2011, http://scottchacon.com/2011/08/31/github-flow.html.)*

Git-flow



https://nvie.com/posts/a-successful-git-branching-model/

GitHub-flow



https://guides.github.com/introduction/flow/

- Introduction
- GitHub "Pull Request"
- **Dangers des processus d'approbation de changement**
- Dangers du «contrôle total des changements»
- Coordination et planification des modifications
- Révision par les pairs
- Pair Programming
- Processus de Pull Request
- Élimination des processus bureaucratiques
- Conclusion
- Conclusion : Partie IV

Dangers des processus d'approbation de changement

- L'échec de Knight Capital est l'une des erreurs de déploiement de logiciel les plus importantes de la mémoire récente
  - Une **erreur de déploiement de quinze minutes a entraîné une perte de 440 millions de dollars** au cours de laquelle les équipes d'ingénierie ont été incapables de désactiver les services de production
  - Les pertes financières ont mis en péril les activités de l'entreprise et ont forcé celle-ci à être vendue au cours du week-end afin qu'elles puissent continuer à fonctionner sans mettre en péril tout le système financier

Dangers des processus d'approbation de changement

- John Allspaw (Adaptive Capacity Labs, Etsy) a observé que lorsque des incidents graves se produisent, tels que l'accident de déploiement de Knight Capital, **deux explications contrefactuelles expliquent pourquoi l'accident s'est produit**
  - Le premier récit est que l'accident était dû à une **défaillance du contrôle des modifications**.
    - ce qui semble valable parce que nous pouvons imaginer une situation dans laquelle de meilleures pratiques de contrôle des modifications auraient pu détecter le risque plus tôt et empêcher le changement d'entrer en production
    - Et si nous ne pouvions pas l'empêcher, nous aurions peut-être pris des mesures pour permettre une détection et une récupération plus rapides
  - Le deuxième récit est que l'accident était dû à un **échec des tests**
    - Cela semble également valable, car avec de meilleures pratiques de test, nous aurions pu identifier le risque plus tôt et annuler le déploiement à risque, ou au moins nous aurions pu prendre des mesures pour permettre une détection et une récupération plus rapides

Dangers des processus d'approbation de changement

- La réalité surprenante est que, dans les environnements où règnent une culture de méfiance et de commande-et-contrôle (command-and-control), les résultats de ces types de contrôle du changement et les contre-mesures de test entraînent souvent une probabilité accrue que des problèmes se reproduisent, avec des résultats encore pires

- Gene Kim (co-auteur de ce livre) décrit sa réalisation:
  - le changement et les contrôles de tests peuvent potentiellement avoir l'effet inverse de celui souhaité:

    «L'un des moments les plus importants de ma carrière professionnelle. Ce moment 'aha' était le résultat d'une conversation en 2013 avec John Allspaw et Jez Humble à propos de l'accident de Knight Capital, me faisant remettre en question certaines de mes convictions fondamentales formées au cours des dix dernières années, notamment après avoir suivi une formation d'Auditeur. »

Dangers des processus d'approbation de changement

- Il a ajouté:

  «Si bouleversant que cela ait été, ce fut aussi un moment très instructif pour moi.

  Non seulement ils m'ont convaincu qu'ils avaient raison, mais nous avons également testé ces convictions dans le rapport 2014 sur l'état de DevOps,

  qui a abouti à des découvertes étonnantes qui confirment que **la création d'une culture de confiance élevée constitue probablement le défi de gestion le plus important de cette décennie.** »

- Introduction
- GitHub "Pull Request"
- Dangers des processus d'approbation de changement
- **Dangers du «contrôle total des changements»**
- Coordination et planification des modifications
- Révision par les pairs
- Pair Programming
- Processus de Pull Request
- Élimination des processus bureaucratiques
- Conclusion
- Conclusion : Partie IV

Dangers du «contrôle total des changements»

- Les contrôles de changement traditionnels peuvent conduire à des résultats inattendus, tels que de longs délais d'exécution, et la réduction de la force et de l'immédiateté du retour d'informations généré par le processus de déploiement
- Afin de comprendre comment cela se produit, examinons les contrôles que nous mettons souvent en place en cas d'échec du contrôle des modifications:
  - **Ajouter plus de questions** auxquelles il faut répondre dans le formulaire de demande de changement
  - **Exiger plus d'autorisations**, telles qu'un niveau d'approbation de la direction supplémentaire (par exemple, au lieu de simplement devoir faire approuver par le vice-président des opérations, nous exigeons maintenant que le directeur informatique approuve également) ou davantage de parties prenantes (par exemple, ingénierie réseau, conseils de révision d'architecture, etc.)
  - **Exiger plus de temps pour l'approbation** des modifications afin que les demandes de modification puissent être correctement évaluées

Dangers du «contrôle total des changements»

- Ces contrôles ajoutent souvent plus de friction au processus de déploiement en multipliant le nombre d'étapes et d'approbations, et en augmentant la taille des lots et les délais de déploiement
  - Ce qui, nous le savons, réduit les chances de succès de la production pour Dev et Ops
  - Ces contrôles réduisent également la rapidité avec laquelle nous obtenons des informations en retour de notre travail
- L'une des croyances fondamentales du système de production Toyota est que «**les personnes les plus proches d'un problème sont celles qui en savent généralement le plus**»
  - Cela devient d'autant plus prononcé que le travail exécuté et le système dans lequel il est exécuté deviennent plus complexes et dynamiques, tel que dans les flux de valeur typiques en DevOps
  - Dans ces cas, la création d'étapes d'approbation par des personnes situées de plus en plus loin du travail peut en réalité réduire les chances de réussite
  - Comme il a été prouvé à maintes reprises, **plus la distance entre la personne effectuant le travail** (i.e. le responsable du changement) **et celle qui décide de le faire** (i.e. l'autorisateur du changement) **est grande**, **plus le résultat sera mauvais**

Dangers du «contrôle total des changements»

- Dans le Puppet Labs' 2014 State of DevOps Report, l'une des principales conclusions est que **les organisations les plus performantes s'appuient davantage sur l'examen par les pairs et moins sur l'approbation externe des modifications**
- La figure 41 montre que plus les entreprises s'appuient sur les approbations de modification, plus leurs performances informatiques sont dégradées en termes de stabilité (délai moyen de restauration du service et taux d'échec de la modification) et de débit (délais de déploiement, fréquence de déploiement)
- Dans de nombreuses organisations, les comités consultatifs de changement jouent un rôle important dans la coordination et la gouvernance du processus de livraison, mais leur travail ne devrait pas consister à évaluer manuellement tous les changements, et ITIL ne le recommande pas

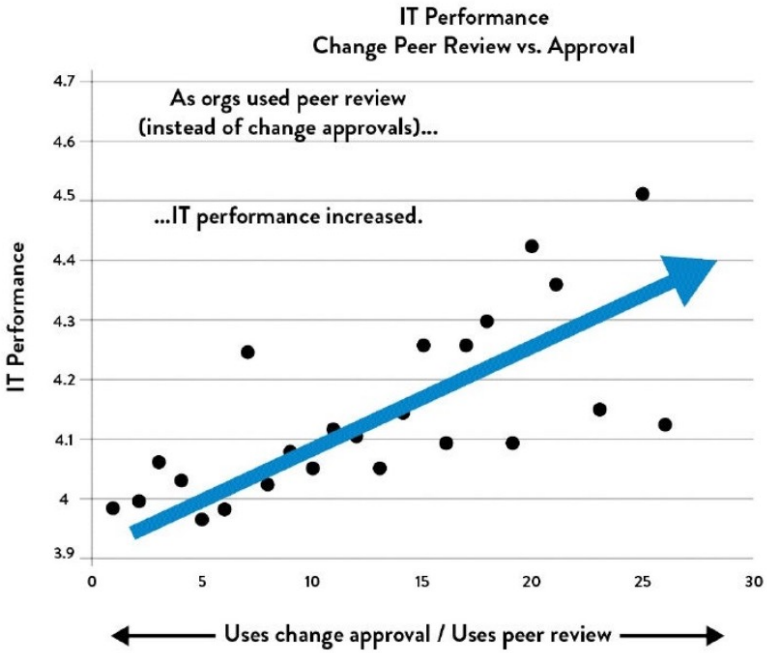


Figure 41: *Organizations that rely on peer review outperform those with change approvals (Source: Puppet Labs, DevOps* Survey Of Practice 2014*)*

Dangers du «contrôle total des changements»

- Pour comprendre pourquoi il en est ainsi, considérons la situation délicate d'une personne qui siège sur un comité consultatif de modification, et qui doit passer en revue une modification complexe composés de centaines de milliers de lignes de code, créé par des centaines d'ingénieurs
  - À un extrême, nous ne pouvons pas prédire de manière fiable si un changement aboutira, soit en lisant une description du changement en cent mots, soit en validant simplement qu'une liste de contrôle a été complétée
  - À l'autre extrême, il est peu probable que l'analyse minutieuse de milliers de lignes de modifications de code révèle de nouvelles informations. Une partie de ceci est la nature de faire des changements à l'intérieur d'un système complexe. Même les ingénieurs qui travaillent à l'intérieur de la base de code dans le cadre de leur travail quotidien sont souvent surpris par les effets secondaires de changements qui devraient être à faible risque

Dangers du «contrôle total des changements»

- Pour toutes ces raisons, nous **devons créer des pratiques de contrôle efficaces** qui s'apparentent davantage à un **examen par les pairs**, réduisant ainsi notre recours à des organismes externes pour autoriser nos changements
- Nous devons également **coordonner et programmer efficacement les changements**
- Nous explorons ces deux aspects dans les deux prochaines sections

- Introduction
- GitHub "Pull Request"
- Dangers des processus d'approbation de changement
- Dangers du «contrôle total des changements»
- **Coordination et planification des modifications**
- Révision par les pairs
- Pair Programming
- Processus de Pull Request
- Élimination des processus bureaucratiques
- Conclusion
- Conclusion : Partie IV

Coordination et planification des modifications

- Lorsque plusieurs groupes travaillent sur des systèmes partageant des dépendances, nos modifications devront probablement être coordonnées pour éviter toute interférence les unes par rapport aux autres
  - E.g. le marshaling, le traitement par lots et la séquence des modifications
- En général, **plus notre architecture est faiblement couplée**, **moins nous avons besoin de communiquer et de nous coordonner** avec les autres équipes de composants
  - Lorsque l'architecture est véritablement orientée service, les équipes peuvent effectuer des modifications avec un degré élevé d'autonomie, là où les modifications locales ont peu de chance de créer des perturbations globales
- Toutefois, même dans une architecture à faible couplage, lorsque de nombreuses équipes effectuent des centaines de déploiements indépendants par jour, des modifications peuvent interférer (par exemple, des tests A/B simultanés)
- **Pour atténuer ces risques**, nous pouvons **utiliser des "chat rooms"** pour annoncer les modifications et rechercher de manière proactive les collisions pouvant exister

Coordination et planification des modifications

- **Pour les organisations plus complexes et les organisations avec des architectures à couplage plus étroit**, il **peut être nécessaire de planifier délibérément nos modifications**, où les représentants des équipes se réunissent, non pas pour autoriser les modifications, mais pour planifier et ordonner leurs modifications afin de minimiser les accidents
- Cependant, **certains domaines**, tels que les modifications d'infrastructure globale (par exemple, les changements de commutateur de réseau central), **seront toujours associés à un risque plus élevé**
  - Ces modifications nécessiteront toujours des contre-mesures techniques, telles que la redondance, le basculement, les tests complets et (idéalement) la simulation

- Introduction
- GitHub "Pull Request"
- Dangers des processus d'approbation de changement
- Dangers du «contrôle total des changements»
- Coordination et planification des modifications
- **Révision par les pairs**
- Pair Programming
- Processus de Pull Request
- Élimination des processus bureaucratiques
- Conclusion
- Conclusion : Partie IV

Révision des modifications par les pairs

- **Au lieu de requérir l'approbation d'un organisme externe, nous pouvons demander aux ingénieurs d'obtenir des examens de leurs modifications par des pairs**
  - En développement, cette pratique a été appelée révision de code, mais elle **s'applique également à toute modification apportée à nos applications ou environnements**, e.g. serveurs, réseaux et BD
  - Objectif: **détecter les erreurs en demandant à nos collègues impliqués dans le travail effectué de réviser nos changements**
  - **Améliore la qualité des changements**, en plus de crée de la **formation croisée**, de l'**apprentissage par les pairs** et de l'**amélioration des compétences**

- Il est logique de demander des révisions avant de soumettre le code dans le tronc du contrôle de source
  - Au minimum, nos collègues ingénieurs devraient examiner notre modification
  - Pour les domaines à risque plus élevé (e.g. modifications de BD ou de composants stratégiques avec une couverture de test automatisée insuffisante), un expert en la matière peut être amené à examiner les modifications ou on peut organiser des révisions additionnelles (e.g «+2» au lieu de «+1»)

Examen des modifications par les pairs

- Le principe des lots de petite taille s'applique également aux révisions de code
  - Plus l'ampleur du changement à examiner est importante, plus la compréhension est longue et longue, plus le fardeau de l'ingénieur examinateur est lourd
  - Comme Randy Shoup l'a fait remarquer:

    «Il existe une relation non linéaire entre l'ampleur du changement et le risque potentiel d'intégration de ce changement: lorsque vous passez d'un changement de code de dix lignes à un code de 100 lignes, le risque d'erreur est plus de dix fois plus élevé, et ainsi de suite. ”
  - C'est pourquoi il est essentiel que les développeurs travaillent par petites étapes incrémentales plutôt que sur des branches de fonctionnalités de longue durée

- En outre, notre capacité à critiquer de manière significative les modifications de code diminue à mesure que la taille des modifications augmente
  - Comme Giray Özil l’a tweeté:

    «Demandez à un programmeur d’examiner dix lignes de code, il trouvera dix problèmes. Demandez-lui de faire cinq cents lignes et il dira que ça a l'air bien. »

Examen des modifications par les pairs

- **Les instructions pour la révision du code incluent**:
  - Tout le monde doit avoir quelqu'un pour réviser ses modifications (par exemple, le code, l'environnement, etc.) avant de soumettre sur le tronc ("commit to trunk")
  - Tout le monde devrait surveiller le flux de validation des membres de son équipe afin que les conflits potentiels puissent être identifiés et examinés
  - Définissez les modifications considérées comme présentant un risque élevé et pouvant nécessiter l'examen d'un expert en la matière (modifications à une base de données, modules sensibles à la sécurité, tels que l'authentification, etc.)
  - Si quelqu'un soumet un changement trop volumineux pour être réviser facilement - autrement dit, vous ne pouvez pas comprendre son impact après l'avoir lu plusieurs fois, ou vous devez demander des éclaircissements à l'auteur de la communication - il faut le scinder en deux, plusieurs changements plus petits qui peuvent être compris en un coup d'œil
- Pour nous assurer qu'il ne s'agit pas simplement d'examens d'estampes ("rubber stamping reviews"), nous pouvons également vouloir examiner les statistiques d'examens de code pour déterminer le nombre de modifications proposées approuvées ou non approuvées, et éventuellement échantillonner et inspecter des examens de codes spécifiques

Examen des modifications par les pairs

- Les revues de code se présentent sous différentes formes:
  - **Programmation en binôme ("Pair programming"):** les programmeurs travaillent en binôme (voir section ci-dessous)
  - **"Au-dessus de l'épaule » ("Over-the-shoulder"):** un développeur regarde par-dessus l'épaule de l'auteur lorsque celui-ci parcourt le code
  - **Transmission de courriers électroniques:** un système de gestion de code source envoie automatiquement le code aux réviseurs après son enregistrement ("checked in")
  - **Révision de code assistée par outil:** les auteurs et les relecteurs utilisent des outils spécialisés conçus pour la révision de code par des pairs (Gerrit, pull requests GitHub, merge requests GitLab, etc.) ou des outils fournies par les référentiels de code source (GitHub, Mercurial, Subversion, et plates-formes telles que Gerrit, Atlassian Stash et Atlassian Crucible)
- Un examen minutieux des modifications sous de nombreuses formes est efficace pour localiser les erreurs précédemment négligées
- Les révisions de code peuvent faciliter l'augmentation des mises à jour de code et des déploiements en production, et prendre en charge le déploiement basé sur les lignes réseau et la diffusion continue à grande échelle, comme nous le verrons dans l'étude de cas suivante

Étude de cas : Google (2010)

- La figure 42 montre comment les délais de révision du code sont affectés par la taille du changement
  - L'axe des abscisses représente la taille de la modification, et l'axe des ordonnées, le délai nécessaire au processus de révision du code
  - En général, plus le changement soumis pour les révisions de code est important, plus le délai requis pour obtenir les approbations nécessaires est long
  - Et les points de données dans le coin supérieur gauche représentent les changements plus complexes et potentiellement risqués qui ont nécessité plus de délibération et de discussion

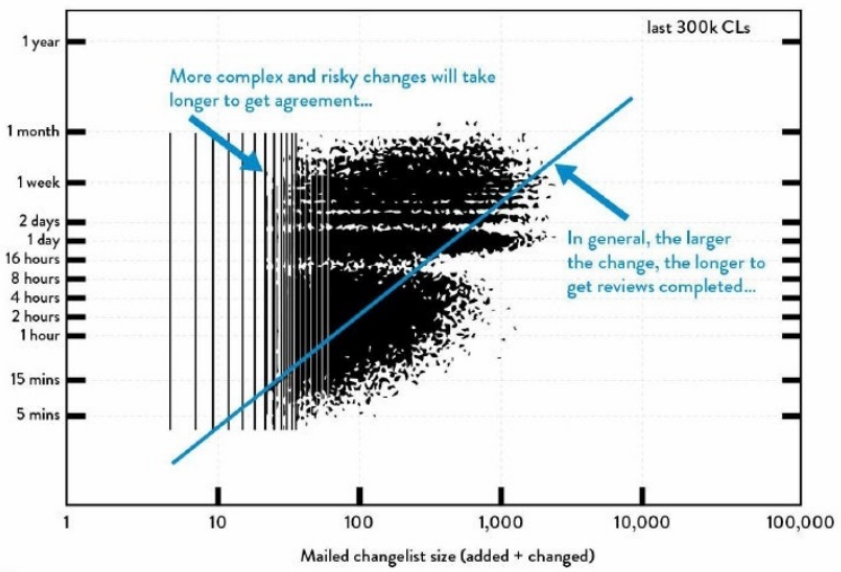
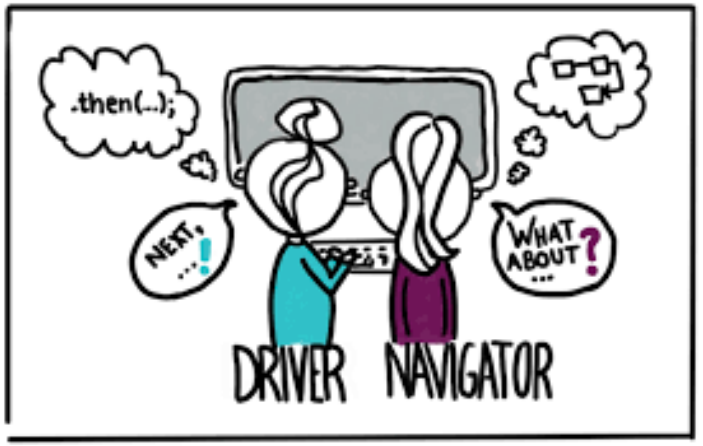


Figure 42: *Size of change vs. lead time for reviews at Google (Source: Ashish Kumar, "Development at the Speed and Scale of Google," presentation at QCon, San Francisco, CA, 2010, https://qconsf.com/sf2010/dl/qcon-sanfran-2010/slides/AshishKumar_DevelopingProductsattheSpeedandScaleofGoogle.pdf.)*

- Introduction
- GitHub "Pull Request"
- Dangers des processus d'approbation de changement
- Dangers du «contrôle total des changements»
- Coordination et planification des modifications
- Révision par les pairs
- Dangers de faire plus de tests manuels
- **Pair Programming**
- Processus de Pull Request
- Élimination des processus bureaucratiques
- Conclusion
- Conclusion : Partie IV

Pair Programming

- La programmation en binôme (Pair programming)
  - Deux ingénieurs qui travaillent ensemble sur le même poste de travail
  - Méthode popularisée par Extreme Programming (XP) et Agile au début des années 2000
  - Comme pour les révisions de code, cette pratique a commencé dans Développement mais s'applique également au travail effectué par n'importe quel ingénieur dans notre chaîne de valeur
  - Dans ce livre, nous utiliserons le terme appariement ("pairing") et programmation en binôme ("pair programming") de manière interchangeable, pour indiquer que la pratique ne concerne pas uniquement les développeurs



Pair Programming

- Dans un schéma d'appariement commun
  - Un ingénieur remplit le rôle de conducteur ("driver"), la personne qui écrit réellement le code
  - Tandis que l'autre ingénieur joue le rôle de navigateur, d'observateur ou de pointeur, la personne qui passe en revue le travail exécuté
  - L'observateur peut également examiner l'orientation stratégique des travaux, en proposant des idées d'amélioration et les problèmes potentiels à résoudre
  - Permet au conducteur de concentrer toute son attention sur les aspects tactiques de la réalisation de la tâche, en utilisant l'observateur comme filet de sécurité et guide
  - Lorsque les deux spécialités sont différentes, les compétences sont transférées comme un effet secondaire automatique, que ce soit par le biais d'une formation ad-hoc ou par le partage de techniques et de solutions de contournement
- Un autre modèle de programmation en binôme renforce le développement piloté par les tests (TDD) en demandant à un ingénieur d'écrire le test automatisé et à l'autre ingénieur d'appliquer le code

Pair Programming

- Dre Laurie Williams a réalisé une étude en 2001 qui montrait que

  «**les programmeurs jumelés ralentissent de 15%** par rapport à deux programmeurs indépendants, tandis que **le code "sans erreur" augmente de 70% à 85%**.

  Étant donné que les tests et le débogage sont souvent beaucoup plus coûteux que la programmation initiale, il s'agit d'un résultat impressionnant.

  Les paires **considèrent généralement plus de solutions de conception** que les programmeurs travaillant seuls et **aboutissent à des conceptions plus simples et plus faciles à gérer**;

  elles **détectent également les défauts de conception à un stade précoce**. »

- La Dre Williams a également signalé que 96% de ses répondants ont déclaré aimer davantage leur travail lorsqu'ils programment en binôme que lorsqu'ils programment seul
- La programmation en pair présente l'avantage supplémentaire de **diffuser les connaissances au sein de l'organisation et d'augmenter le flux d'informations au sein de l'équipe**
  - Faire réviser par des ingénieurs plus expérimentés, alors que les codes d'ingénieur moins expérimentés est également un moyen efficace d'enseigner et d'être enseigné

- Introduction
- GitHub "Pull Request"
- Dangers des processus d'approbation de changement
- Dangers du «contrôle total des changements»
- Coordination et planification des modifications
- Révision par les pairs
- Dangers de faire plus de tests manuels
- Pair Programming
- **Processus de Pull Request**
- Élimination des processus bureaucratiques
- Conclusion
- Conclusion : Partie IV

Efficacité des processus de Pull Request

- Une autre méthode vient de Ryan Tomayko (CIO et cofondateur de GitHub et l'un des inventeurs du processus de Pull Request)
  - Lorsqu'on lui a demandé de décrire la différence entre une mauvaise et une bonne Pull Request, il a répondu que cela n'avait pas grand-chose à voir avec le résultat de la production
  - Une Pull Request incorrecte est une demande qui de fournie pas assez de contexte pour le lecteur, elle ne contient que peu ou pas de documentation sur le but recherché par le changement
  - Par exemple, une pull request contenant simplement le texte suivant: «Résolution du problème n ° 3616 et n ° 3841.» **
    - C'était une Pull Request de GitHub, critiquée par Tomayko:

      «C'est probablement un nouvel ingénieur qui a écrit ici.
    - Tout d'abord, aucun ingénieur en particulier n'a été spécifiquement mentionné - au minimum, l'ingénieur aurait dû mentionner son mentor ou un expert en la matière dans le domaine qu'il modifie pour faire en sorte que l'opérateur concerné révise son changement.
    - Pire encore, rien n'explique ce que sont réellement les changements, pourquoi ils sont importants ou exposent les idées de celui qui les met en œuvre. »

Efficacité des processus de Pull Request

- D'autre part, lorsqu'on lui a demandé de décrire le schéma d'une bonne (excellente) Pull Request qui représente bien un processus de contrôle efficace, Tomayko a rapidement énuméré les éléments essentiels
  - **Il doit y avoir suffisamment d'information par rapport aux points suivants**:
    - **Raisons** pour lesquelles le changement a été apporté
    - **Comment** le changement a été fait
    - Identification des **risques** et **contre-mesures associées**
  - Tomayko recherche également une bonne discussion sur le changement, rendue possible par tout le contexte associé à la Pull Request:
    - Souvent, des risques supplémentaires sont signalés, des idées sur les meilleurs moyens de mettre en œuvre le changement souhaité, des idées sur la manière de mieux atténuer les risques, et ainsi de suite
    - Et si quelque chose de mauvais ou d'inattendu se produit lors du déploiement, il est ajouté à la Pull Request, avec un lien vers le problème correspondant
    - Toute discussion se passe sans blâmer; au lieu de cela, il y a une conversation franche sur la façon d'éviter que le problème ne se reproduise à l'avenir

- Introduction
- GitHub "Pull Request"
- Dangers des processus d'approbation de changement
- Dangers du «contrôle total des changements»
- Coordination et planification des modifications
- Révision par les pairs
- Dangers de faire plus de tests manuels
- Pair Programming
- Processus de Pull Request
- **Élimination des processus bureaucratiques**
- Conclusion
- Conclusion : Partie IV

Élimination des processus bureaucratiques

- Jusqu'à présent, nous avons discuté de processus de révision par les pairs et de programmation en paires nous permettant d'améliorer la qualité de notre travail sans recourir à des approbations externes pour les modifications
- Cependant, de nombreuses entreprises ont encore des processus d'approbation de longue date qui nécessitent des mois de navigation
- Ces processus d'approbation peuvent augmenter considérablement les délais, non seulement pour nous empêcher de fournir de la valeur rapidement aux clients, mais également pour accroître les risques pour nos objectifs organisationnels
- Lorsque cela se produit, nous devons repenser nos processus pour pouvoir atteindre nos objectifs plus rapidement et en toute sécurité.

- Introduction
- GitHub "Pull Request"
- Dangers des processus d'approbation de changement
- Dangers du «contrôle total des changements»
- Coordination et planification des modifications
- Révision par les pairs
- Dangers de faire plus de tests manuels
- Pair Programming
- Processus de Pull Request
- Élimination des processus bureaucratiques
- **Conclusion**
- Conclusion : Partie IV

Conclusion

- Dans ce chapitre, nous avons discuté de la manière d'intégrer dans notre travail quotidien des pratiques qui améliorent la qualité de nos changements et réduisent le risque de résultats de déploiement médiocres, réduisant ainsi notre dépendance à l'égard des processus d'approbation
  - Les études de cas de GitHub et Target montrent que ces pratiques non seulement améliorent nos résultats, mais réduisent également considérablement les délais d'exécution et augmentent la productivité des développeurs
  - Faire ce genre de travail nécessite une culture de grande confiance
- Pensez à l'histoire racontée par John Allspaw à propos d'un ingénieur débutant:
  - L'ingénieur a demandé s'il était correct de déployer un petit changement HTML, et Allspaw a répondu: «Je ne sais pas, qu'en penses-tu?»
  - Il a ensuite demandé:
    - «Avez-vous demandé à quelqu'un d'examiner votre changement?
    - Savez-vous qui est la meilleure personne à qui demander pour des modifications de ce type?
    - Avez-vous fait tout ce qui était en votre pouvoir pour vous assurer que ce changement fonctionne dans la production telle que conçue?
    - Si vous l'avez fait, alors ne me demandez pas: faites simplement le changement! "

Conclusion

- En réagissant de la sorte, Allspaw a rappelé à l'ingénieur qu'elle était seule responsable de la qualité de son changement
  - Si elle faisait tout ce qu'elle pouvait pour être sûre que le changement fonctionnerait, elle n'aurait alors pas besoin de demander l'approbation de qui que ce soit. Elle devrait simplement faire le changement
- La création des conditions permettant aux personnes implémentant un changement de s'approprier pleinement la responsabilité de la qualité de leurs changements est un élément essentiel de la culture générative de haute confiance que nous nous efforçons de créer
- En outre, ces conditions nous permettent de créer un système de travail de plus en plus sûr, dans lequel nous nous aidons mutuellement à atteindre nos objectifs, en dépassant toutes les limites nécessaires pour y parvenir

- Introduction
- GitHub "Pull Request"
- Dangers des processus d'approbation de changement
- Dangers du «contrôle total des changements»
- Coordination et planification des modifications
- Révision par les pairs
- Dangers de faire plus de tests manuels
- Pair Programming
- Processus de Pull Request
- Élimination des processus bureaucratiques
- Conclusion
- **Conclusion : Partie IV**

Conclusion : Partie IV

- La quatrième partie nous a montré qu'en mettant en place des boucles de rétroaction, nous permettons à tout le monde de
  - Travailler ensemble pour atteindre des objectifs communs
  - Voir les problèmes au fur et à mesure qu'ils se présentent, et
  - Grâce à une détection et à une récupération rapides, de garantir que les fonctionnalités fonctionnent non seulement comme prévu en production, mais qu'elles permettre également d'atteindre les objectifs organisationnels et l'apprentissage organisationnel
- Nous avons également examiné comment établir des objectifs partagés couvrant les activités de développement et d'exploitation afin qu'ils puissent améliorer la santé de l'ensemble du flux de valeur
- Nous sommes maintenant prêts à entrer dans la Partie V: La Troisième Voie, Les Pratiques Techniques de l'apprentissage, afin de créer des opportunités d'apprentissage qui se produisent plus tôt et de plus en plus rapidement et à moindre coût, et de susciter une culture d'innovation et d'expérimentation permettant à chacun de faire un travail significatif qui aide notre organisation à réussir