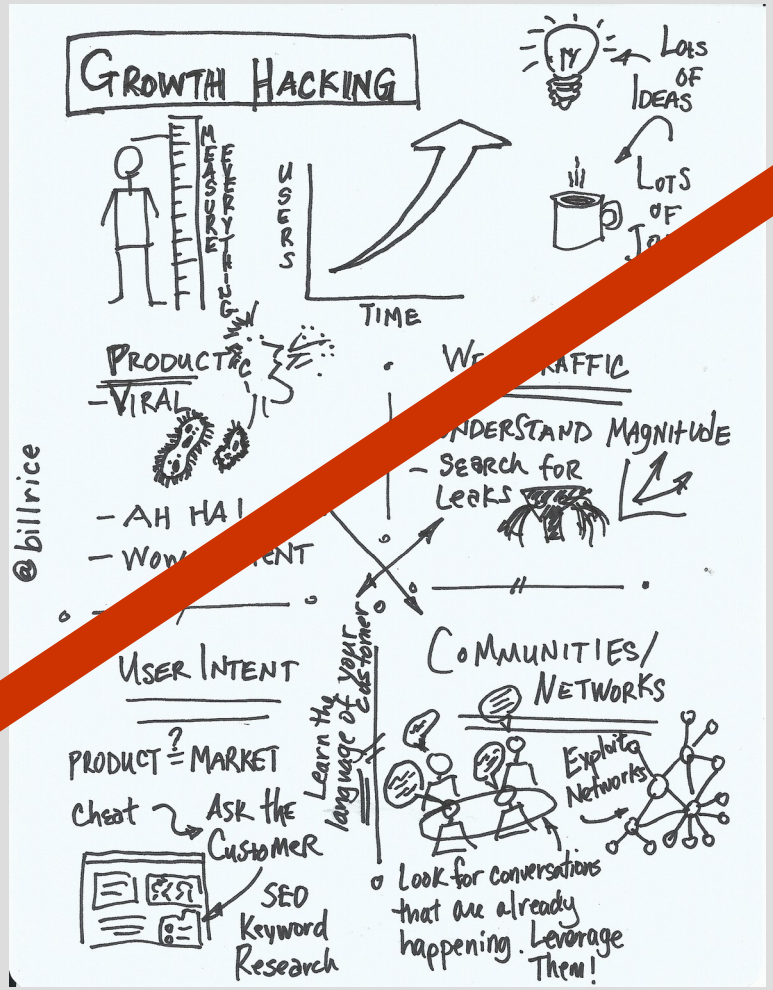
Coder pour des millions de personnes

@vcasse

Tu as codé pour des millions de personnes ?

Qui veut gagner des millions ?



Statistiques

	1	**10**				
1 %	1 chance sur **100**	1 chance sur **10**				

Statistiques

	1	**10**	**100**	**1 000**		
1 %	1 chance sur **100**	1 chance sur **10**	**1**	**10**		

Statistiques

	1	10	100	1 000	100 000	1 000 000
1 %	1 chance sur **100**	1 chance sur **10**	**1**	**10**	**1 000**	**10 000**

Statistiques

	1	**10**	**100**	**1 000**	**100 000**	**1 000 000**
1 %	1 chance sur **100**	1 chance sur **10**	**1**	**10**	**1 000**	**10 000**
0,1 %	1 chance sur **1 000**	1 chance sur **100**	1 chance sur **10**	**1**	**100**	**1000**

Statistiques

	1	**10**	**100**	**1 000**	**100 000**	**1 000 000**
1 %	1 chance sur **100**	1 chance sur **10**	**1**	**10**	**1 000**	**10 000**
0,1 %	1 chance sur **1 000**	1 chance sur **100**	1 chance sur **10**	**1**	**100**	**1000**
0,001 %	1 chance sur **100 000**	1 chance sur **10 000**	1 chance sur **100**	1 chance sur **10**	**1**	**10**

Statistiques

	1	10	100	1 000	100 000	1 000 000
1 %	1 chance sur **100**	1 chance sur **10**		**10**	**1 000**	**10 000**
0,1 %	1 chance sur **1 000**	1 chance sur **100**	1 chance sur **10**	**1**	**100**	**1000**
0,001 %	1 chance sur **100 000**	1 chance sur **10 000**	1 chance sur **100**	1 chance sur **10**	**1**	**10**

RISQUE

Impact

Définit selon :

- Effet sur le produit et sur les clients
- Temps de reprise sur activité
- Difficulté de diagnostic

Exemples :
- Impact faible : panne d’API
- Impact fort : panne de serveur web
- Impact catastrophique : bug qui altère des données

Gérer le risque au quotidien

Risque x Impact = Criticité

« Si çà passe sur les hébergements web,
c’est bon pour la prod »

Bonnes pratiques de code

pour millionnaires

♥ LEGACY ♥

Base de code ancienne ?

Adoptez un legacy

Le code, c’est la vie

« J’ai ajouté ce paramètre pour gagner du temps si on développe cette fonctionnalité ! »

« J’ai déplacé ce code dans une librairie au cas où on le réutiliserait »

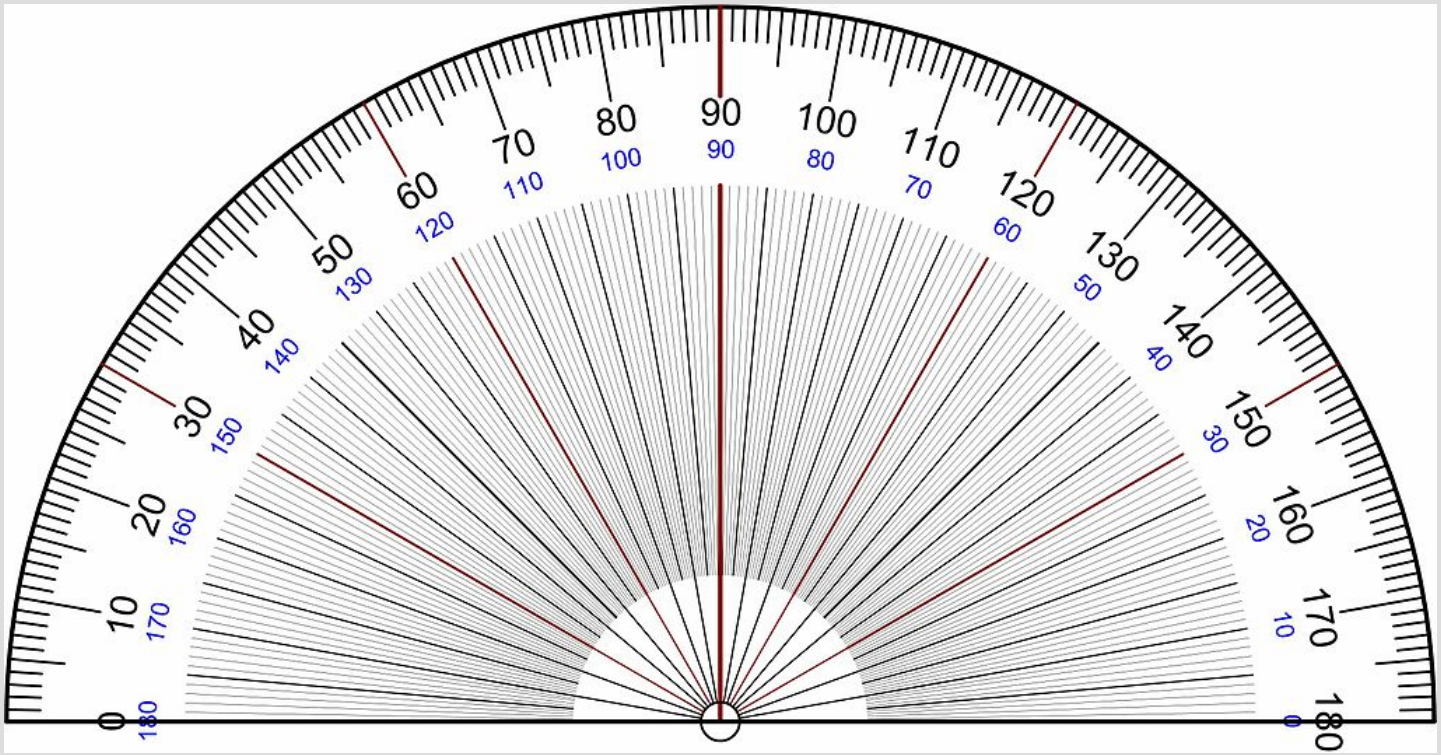
Ne pas faire défaut

- Souple sur les paramètres
- Paramètre obligatoire → risque d'erreurs
- Remontez explicitement les erreurs OU proposez une valeur par défaut

```
function toto( $param, $argument)
{
    if ( !isset( $argument ) ) return 'Missing argument' ;

    if ( !isset ( $param ) ) $param = 'default' ;
    ...
}
```

Oui, tout rapporter est une bonne idée



Remonter toutes les erreurs

- « Çà n’arrivera jamais »
- Intercepter toutes les erreurs
- Monitorer les erreurs
- Aucune confiance dans le code, les infras, les logiciels...
- Structure de données Result
  https://doc.rust-lang.org/rust-by-example/error/result.html

Erreurs récurrentes

- Erreur probable = masse d’erreurs
- Automatiser actions récurrentes
- Fixer les causes racine
  - Mauvais paramètre client ? UX
  - Soucis dans le logiciel ? Adapter le

Et pour les autres erreurs ?

- Outillage manuel
- Déléguer
- Relancer le logiciel (idempotent)

Simplifier le code : restez statique

Avez vous besoin

d’une base de données

pour le faire ?

Organisation au quotidien

- Fixer les erreurs prend du temps
- Fixer les causes racine prend du temps
- Usage massif = tas d’erreurs
- **On doit donc gérer une production**
  et développer de nouveaux projets en parallèle

Infra ?
industrie ?

Ou les deux ?

Quelques chiffres

15 000 serveurs actifs

1 millions de bases de données

5 millions de sites web

40 Gb/s de trafic

Industriel de père en fils

- Les datacentres = industrie
- Gestion de configurations
- CMDB
- Préprovisionning

Monitoring

- Configuration automatique
- Self – healing
- Monitoring du self-healing

Diviser pour mieux gérer

Méthode de déploiement

- Comment tester les niveaux de risque ?
- Déployer par pallier : 1 / 10 / 100 / 1000 / all

Gestion multi - versions

- Plusieurs versions en // sur l’infrastructure
- Le code doit évoluer par mise à jour successives et savoir vivre avec ces multiples versions

RIP

- Mort audieuse : remonter une erreur si l’infra retourne une erreur
- Mort gracieuse : en cas d’erreur, passer à la suite sans faire de bruit (mais des logs quand même)

Architecture

Micro service by design

- API doit être rapide pour scale
- L’infra prend du temps
- Sécurité

Tout asynchrone

- Mécanisme de « jobs »
- Machine à état + chaînage d’étapes idempotentes

Pas très UX les jobs

- Représentation de l’infra client en BDD → très rapide
- Avec un état et un lien vers le job
- État synchrone d’une action asynchrone

Avez vous des millions d’oreilles ?

Quelques dizaines de paires d’oreilles

- Écoutez votre support
- Donc … ayez un support
- « Process » de feedbacks + cafés

Signaux faibles

- Gardez un œil sur Twitter (même si vous avez un support)
- Lisez les forums
- Les mailings list marchent toujours bien (IRC aussi)

Bibliographie

- Photos :
  - Home https://pxhere.com/fr/photo/563721
  - Growth hacking https://www.flickr.com/photos/billrice/9683037300
  - Forêt :
  https://fr.m.wikipedia.org/wiki/Fichier:Retjons_(Landes,_Fr),_chemin_en_for%C3%AAt.JPG

Des questions ?

Merci

@vcasse