Le code source comme document d'infrastructure: le cas de Yandex

Journées du Centre Internet & Société - 01.10.2024

Bonjour, je m'appelle Pierre Depaz, et je suis actuellement enseignant au département arts & médias de NYU Berlin, et j'ai fini une thèse l'année dernière en littérature comparée sur la compréhension et l'esthétique du code source, qui se rattache aux *software studies*, et plus spécifiquement, celui des *critical code studies*. En général, je m'intéresse à la manière dont les codes sources sont des documents témoins de l'inscription d'entités diverses dans un environnement numérique, et donc à la manière dont ils sont le résultat d'une opération de traduction, entre concepts humains et langages machines.

Un des corpus sur lequel je travaille en ce moment, c'est le code source de Yandex, le géant informatique russe. C'est un travail qui est encore au stade de recherches exploratoire—on comprendra vite pourquoi—, donc aujourd'hui, je vais surtout m'attacher à vous présenter rapidement une méthodologie, un document, quelques découvertes préliminaires, et puis surtout conclure sur des questionnements encore ouverts.

Méthodologie générale - 1min

Pour commencer, un peu de contexte méthodologique. Les *Critical Code Studies* vont chercher à appliquer des techniques d'analyses littéraires à des objets numériques. Plutôt que de l'application de techniques épistémologiques numériques à des corpus d'humanités, il s'agit plutôt d'appliquer des techniques épistémologiques d'humanités à des corpus numériques.

Cette approche se repose donc sur l'interprétation de codes sources, afin de mettre à jour les présumés, les manques, les connections et les résonnances entre signes et référents. L'écriture de codes sources n'est pas, en effet, uniquement focalisée sur ce que l'ordinateur va comprendre lors de l'interprétation (ou de la compilation) du document. Cette écriture a au moins un autre but: permettre la communication d'humain à humain, entre collègues au sein d'une organisation, par exemple, au sujet d'un problème qui doit être réglé de manière computationnelle. Le code source va donc avoir un sens particulier, ancré dans un contexte social, historique et économique, marqué par une compréhension constamment en évolution de notion d'état, et de fonction. Allant au-delà de sa stricte fonctionnalité, l'idée est qu'il est possible de le considérer comme un discours, et donc avec toutes les connotations, implications, résonances et obfuscation que cela implique.

Depuis une douzaine d'années, et la publication de `10 PRINT CHR$(205.5+RND(1)); : GOTO 10` en 2012, les *critical code studies* se sont donc attelées à expliciter des concepts présents de manière implicite dans les écritures et lectures du code. Nick Montfort et ses collègues se sont emparés des thèmes d'aléatoire, de tissage et de labyrinthes saisissables dans le

one-liner éponyme. D'autre part, Mark Marino a travaillé sur le statut épistémologique des simulations scientifiques et la place des *magic numbers* au sein de celles-ci, tandis que David Berry a pu tracer la notion de *toxicité* telle qu'elle est réifiée dans le code source de "WIT Toxicity Text Model Comparison" (source). Avec d'autres collaboratrices, ces deux derniers travaillent actuellement sur une exégèse du code source d'ELIZA, le premier chatbot. Dans toutes ces études, il s'agit de partir du code pour extraire le rôle de distribution d'agentivité des logiciels.

Un des grands problème du code source, c'est que son accès est très ambivalent. D'une part, le mouvement du *free software* et de l'*open-source* permettent la circulation facile et garantie de textes de programmes. D'autre part, le fait que le code source soit devenu une forme de propriété intellectuelle âprement défendue par des organisations à but lucratifs extraient d'autres textes de programmes de cette circulation ouverte. Entre ces deux modes de circulation, on trouve cependant le *leak*.

Un document - Yandex - 2min

Le 25 Janvier 2023, un nouveau compte utilisateur poste sur le forum *BreachForums* un lien torrent vers un fichier de 44.7 Gb contenant du code source de quasiment tous les services de Yandex.

Il manque à cela toutes les données, les pondérations des modèles d'IA utilisés dans divers services, ainsi que le versionnage des fichiers. D'ailleurs, la première chose que l'on remarque à ce propos, c'est que chaque fichier a la même date de création: le 24 Février 2022, jour de l'invasion massive de l'Ukraine par la Russie, ce qui semble marquer des motivations politiques plutôt qu'économiques de la personne à l'origine du leak.

Dans quel genre de contexte ont donc été écrites ces lignes de codes? Yandex, c'est une entreprise russe privée avec un peu plus de 25 000 employés, un siège administratif aux Pays-Bas, et une relation plus ou moins tumultueuse avec le gouvernement russe. En tant que plateforme numérique, c'est une myriade de service, avec entre autres:

- Browser, un navigateur internet
- Search, un moteur de recherche
- Mail, un service de messagerie
- Eats, un service de livraison de nourriture à domicile
- Metrica, un service d'analyses de données
- Maps, un service de cartes
- Disk, un service de stockage de données
- News, un service d'aggrégation d'actualités

C'est donc non seulement une plateforme numérique, mais un plateforme quasiment hégémonique par l'étendue des services qu'elle propose, mais aussi très significative par ses parts de marché en Russie. Plus de la moitié des recherches Internet se font via Yandex (au niveau mondial, Google est premier avec 90% des recherches). De par cette importance, il a déjà été

étudié par des chercheuses et chercheurs selon les logiques algorithmiques qui vont gouverner l'organisation et la présentation de contenus (e.g. Daucé et Loveluck, 2021).

L'intérêt d'avoir accès au code source, c'est donc de pouvoir inspecter les "coulisses" de cette plateforme, non plus en opérant à travers l'interface client, mais à travers la perspective normative du code source. Il y a déjà eu des analyses techniques, notamment en termes de collection de données et de respect de la vie privée (Kaileigh McCrea, 2023), ainsi que de nombreuses analyses de la manière dont Yandex Search gère le SEO, et plus généralement des analyses de cybersécurité.

Néanmoins, ce corpus reste largement inexploré.

Une méthodologie spécifique - 2min

Le premier problème qui se pose à cette exploration, c'est la taille du corpus. Je mentionnais auparavant le chiffre de 44.7Gb de données, mais uniquement quand les fichiers sont compressés! Jusqu'à maintenant, les critical code studies ont étudié des bouts de code, au maximum d'une centaine de lignes donc toujours moins de 1Mb. Avec Yandex, on saute donc à une échelle toute autre.

- 50 services différents (parmi lesquels il manque notamment Yandex News)
- 14 extensions de language différents
- 1130562 nombre de fichiers différents
- un total de X lignes de code (trouver des comparaisons?)

Généralement, lorsqu'on est confronté une base de code aussi extensive, les recommandations de l'ingénierie logicielle repose sur deux piliers: lire la documentation, et exécuter du code. Dans notre cas, les deux s'avèrent impossible (pas d'accès à la documentation interne, mis à part celle qui est directement dans le code, et pas d'instruction sur les informations, variables et bases de données requises pour exécuter un service).

Donc la démarche que j'utilise ici, c'est de constater *a priori* les différents modules, et de les considérer avant tout comme des entités clairement délimitées. Dans un deuxième temps, il s'agira d'établir les connexions entre les différents modules (notamment, des études préliminaires ont montré que beaucoup des données circulant au sein de l'écosystème Yandex vont à un moment transiter par Metrika, un système d'analyse de données à des fins de profilage publicitaire).

Par module, on peut ensuite opérer une analyse par niveaux, selon la taxonomie de John Cayley (2002).

Au plus haut niveau, celui de la **structure**, il est possible de mettre à jour les délimitations des fonctionnalités, les ontologies du module via ses composants les plus importants, les pratiques de communautés de langages et d'ingénierie, ainsi que l'organisation macro d'un logiciel à travers des graphes de dépendance et des rapports entre fichiers de code source et fichiers externes (tels que `blacklist.txt`). Les modélisations d'entités computationnelles, telles que les définitions

de classes, de *header files* ou de définition de variables globales nous font transitionner vers le niveau inférieur.

Au niveau de la **syntaxe**, la lecture se tourne vers les énoncés qui sont décrits dans le code source, notamment les fonctions, ainsi que les branches conditionnelles, les boucles, tous ces construits qui constituent les gestions des flux de données. C'est là que la sémantique fonctionnelle du code va être la plus apparente, puisque c'est le niveau le plus proche du fonctionnement d'un algorithme.

Enfin, le dernier niveau consiste en celui du **vocabulaire**, c'est à dire des jetons lexicaux spécifiques au choix du languages informatique, qu'ils soient des termes réservés par le langage de programmation (tels que `def` ou `True` dans le langage Python), ou des termes choisis par les équipes d'ingénierie logicielle qui ont co-écrit ces textes. C'est peut-être à ce dernier niveau que l'analyse littéraire se retrouve dans un cadre plus familier, puisqu'il est plus simple de détecter et souligner des connotations culturelles dans le choix des mots (e.g. les références racistes et sexistes parsemées dans le code source, ou le fait qu'il y ait un module qui s'appelle `skynet`, ou que le projet de réseau social s'appelle `socialism`), même si ce choix de mots a peu à voir avec la fonctionnalité du logiciel.

Enfin, pour terminer ce tour d'horizon méthodologique, un mot sur les logiciels utilisés. Afin d'explorer ce corpus, je me suis retrouvé à devoir utiliser des logiciels de traitement de texte en langue naturel pour analyser du language machine (ce qui va donc rendre inutile les questions de fréquence de mots, leur catégorisation, leur collocation, etc.). Mon premier besoin, ça a été de pouvoir chercher des mots-clés qui m'intéresseraient, et de pouvoir les resituer dans le contexte de leurs modules. Pour cela, j'avais commencé à utiliser `open-semantic-search`, qui a plusieurs bons aspects, mais qui est assez lent à l'indexation et à la récupération de données. Une solution plus rapide, c'est l'outil de recherche de code `zoekt` par Google. Et puis enfin, puisqu'il s'agit de faire de l'analyse qualitative, je me suis retrouvé à utiliser l'IDE Visual Studio Code pour inspecter des modules particuliers (VS Code arrête de bien marcher lorsqu'on arrive à 2Gb de données environ).

Des trouvailles - 5min

Pour l'instant, je n'ai exploré que deux modules de Yandex, Yandex Maps et Yandex Search, puisque la lecture et la compréhension de code source est particulièrement chronophage et particulièrement ardue. À ce niveau de la recherche, il s'agissait surtout de voir ce qu'il était possible d'y trouver, dans une perspective de traduction analogue-digitale, c'est-à-dire de trouver des extraits qui vont représenter le monde dans toute sa complexité, et la manière dont les languages machines peuvent gérer et organiser cette complexité.

géopolitique des cartes (question de l'universalité) -> plusieurs examples

Alors, le module `maps`, c'est 230,624 fichiers, avec 504 types de fichiers différents, 1.5 million de lignes de code les cinq plus populaires étant `make`, `cpp`, `h`, `py` et `js`. Et en parcourant ces

fichiers, il y a plusieurs pistes qui m'ont semblé intéressantes.

La première, c'est qu'il y a beaucoup de mentions d'OpenStreetMap (2332), souvent sous l'acronyme `osm`. Par exemple, dans le fichier `maps/renderer/tools/vmap2js/js/main.js:244`, on peut trouver:

```
GEOSEARCH
```

Il y a au moins deux parties du code qui sont externes à Yandex. D'une part, `OpenStreetMap`, donc, un système de cartographie contributif. D'autre part, le `L` provient de Leaflet, une libraire open-source de code qui permet facilite la création de, et l'interaction avec, des cartes numériques. L'ironie de la chose, c'est que Leaflet est créée par Volodymyr Agafonkin, un développeur Ukrainien militant actuellement contre l'invasion russe.

Donc la question que je me pose là, c'est la place de l'open-source et de ces usages vis-à-vis de la géopolique. En plus de ces deux projets, il y a aussi énormément de mentions de TensorFlow, le système d'apprentissage machine de Google. Alors que, certains aspects de la navigation Internet semble se refermer, je me demande donc lesquels restent ouverts.

Un autre exemple d'échange international, c'est cette note dans `contrib/fribidi/ChangeLog.old`:

```
fribidi
```

Encore une fois, de manière assez ironique, `fribidi` est un utilitaire GNU pour supporter les alphabets arabes et hébreus.

Si on suit l'usage des tuiles d'OpenStreetMap, on trouve aussi un module de conversion des données OpenStreetMap en un système interne à Yandex dénommé `YT` (`/maps/garden/modules/osm_to_yt/graph.py:10`), prenant en compte, entre autres, la géométrie des pays:

```
OSM_TO_YT
```

Une fois que ces graphes (principalement des *nodes* et des *edges*), on se trouve néanmoins face à un problème: les frontières sont des concepts disputés. Donc parfois Yandex va devoir gérer des coopérations internationales, comme l'espace Schengen.

Donc, dans `libs/road_graph_import_yt/impl/country_borders.cpp`, on trouve une belle définition de ce qu'est un pays libre:

```
FREE COUNTRIES BORDER
```

La vision ici est donc très pragmatique. Selon les employés de Yandex, le point commun entre l'espace Schengen et Israël et Palestine, c'est l'absence de frontière. Une frontière libre, c'est donc une frontière qui ne prend pas en compte la différence des codes nationaux ISO, et qui peut donc être filtrée par la suite pour accommoder différentes représentations pour le client.

Pourtant, les frontières de l'espace Schengen sont beaucoup moins disputées que celles d'Israël. Il faut bien que Yandex Maps se charge de résoudre ces disputes, de manière automatique, notamment à travers un processus de localization. Dans le fichier `libs/locale/impl/region_groups.cpp`, on peut donc trouver:

```
DISPUTED VIEWS
```

qui va alors organizer la visibilité des frontières selon les points de vue nationaux:

```
DISPUTED GROUP ID
```

Ce qui va ensuite être rendu visible au client notamment via `front/services/regions-service/src/lib/politics.v1.js` et `front/services/regions-service/src/lib/utils.js`, où il s'agit liitéralement de créer une réponse politique:

```
CREATE POLITICS RESPONSE
```

Il y a d'autres territoires qui sont contestés du point de vue d'une utilisation principalement russe, et notamment l'Ukraine et la Crimée, ce qui a des implications géographiques, comme on peut le voir dans le niveau de visibilité permis à certaines régions (ici dans le wiki `wikimap/mapspro/services/mrc/libs/privacy/impl/region_privacy.cpp`):

```
PRIVACY MAP
```

Ici, je suis curieux de savoir pourquoi est-ce que les région *adjacentes* à l'Ukraine ont un niveau de visibilité plus bas, alors qu'il n'y a pas de mention de l'Ukraine elle-même (quelle est la logique?). On note aussi le degré de visibilité d'Israël, qui ne semble pas être ouvert à quelque discussion que ce soit sur le wikimaps.

Enfin, pour conclure cette visite de la géographie selon Yandex, il y a un dernier bout de code intéressant, dans `front/services/maps/src/server/middlewares/localization-`

`middleware.ts`, donc un service qui va agir comme filtre lors d'une requête HTTP, et qui contient notamment la fonction suivante:

```
GET CRIMEA STATUS COOKIE
```

Ce qui implique plusieurs choses. D'une part, le statut de la Crimée est stocké sur un cookie du navigateur client, plutôt que sur une addresse IP. D'autre part, si le cookie peut être utilisé pour déterminer l'apparence d'une carte, il est ici utilisé aussi pour déterminer la langue de la page web qui va être envoyée. Je n'ai pas suivi la circulation du flux de données jusqu'au bout, mais mon hypothèse, c'est que la page renvoyée ne va pas être en Ukrainien.

On voit donc des manifestations concrètes de la manière dont une plateforme numérique va lister et régler des disputes géopolitiques spécifique à sa base d'utilisateurs et d'utilisatrices, en combinant des traductions de jeux de données open-source, établissant des disparitions de frontières purement visuelles, et pas seulement géopolitiques, et en se basant sur ces distinctions faites dans le *back-end* pour moduler des réponses dans le *front-end*. Il y a encore beaucoup de travail pour tracer précisément comment interagissent tous ces éléments, mais ceci n'est qu'une première passe.

Une autre grosse partie du module `maps`, c'est aussi la gestion de requêtes utilisateurs pour ce qui est d'établir des trajectoires idéales de circulation, mais aussi de la gestion des données utilisateurs et de la diffusion de publicités à des moments particuliers d'un trajet.

recherche et actualités

Yandex est aussi particulier en ce qu'il lie étroitement les recherches Internet et la diffusion d'articles extraits de sources journalistiques, à travers le système Yandex.News. Avec un accès au code source de ces systèmes de classification permettrait d'apporter une perspective supplémentaire aux études qui ont déjà été faites sur le fonctionnement de Yandex.News, notamment en ce qu'elles ont été faites selon une logique de *black box*.

Dans le module `search` on trouve donc des fichiers liés à l'algorithme de Zen, notamment dans le document `web/rearrange/zen/zen.cpp`, où se trouvent beaucoup de fonctions qui semblent être impliquées dans la génération des la curation algorithmique. Tracer les différents fils de l'execution de l'algorithme d'organisation des contenus est assez compliqué (il y a 78,225 fichiers dans le module `rearrange`), et je n'ai que touché la surface, mais il y a quelques aspects qui peuvent être mis en avant ici.

```
REARRANGE ORGANIC
```

Il y a plusieurs éléments intéressants ici, c'est d'abord le nom de la fonction: `RearrangeOrganic` implique que l'on fasse une différence entre les résultats organiques et les résultats non-organiques (c'est-à-dire forcés par l'algorithme sans avoir de relation directe

avec la recherche). Ici, ce que fait la première boucle, c'est de trouver le premier document non-organique (`goodDocPos`), avant de placer celui en tête du groupe de documents qui va être retourné au client.

D'autre part, la variable `rearrangeParams` est réutilisée en permanence pour conserver une cohérence des résultats. En essayant de trouver l'origine de ce type de donnée, mais je n'ai pas encore trouvé ce qui constituait exactement ces paramètres.

Le fonctionnement de l'algorithme de Zen est donc assez obscur pour l'instant. En revanche, il est possible de changer de perspective en regardant la manière dont cet algorithme de réarrangement est utilisé dans un cas idéal. Dans le fichier `web/rearrange/lingboost/ut/lingboost_ut.cpp` (et j'imagine `lingboost` être un algorithme de réarrangement par boost de résultats) on a justement ce cas de test:

TEST CASE PUTIN

Et donc là on voit que le test de base consiste à voir si la réorganisation de contenu concernant Poutine fonctionne bien, et particulièrement avec une localization UA (l'autre cas de test concerne les chats…). Une des grosses inconnues, c'est aussi le fichier `qtree` et `qbundle`, qui doit être une représentation des résultats de la recherche, même si pour l'instant je n'en sais trop rien.

Je n'ai pas fini de cartographier la manière dont ces paramètres sont manipulés, parce que c'est particulièrement compliqué à lire, mais en revanche, ces paramètres sont composés de règles. Et autant la complexité vient du fait que ce ces règles soient dynamiques, parfois on tombe aussi sur des règles statiques, comme dans le fichier `web/rearrange/stable/stable_news.cpp`.

HIGH QUALITY NEWSMAKER

Donc là, on voit clairement la priorité des sources journalistiques, en tout cas pour ce qui est du module `web`, et pas du module `news`, même si il y a effectivement un sous-module `news`, ce qui illustre peut-être l'enchevêtrement des recherches webs et recherches d'actualité pour le moteur Yandex.

RANK DOCS

Donc il semble que ce soit la source, plutôt que le sujet des documents, qui soit priorisée. Ce qui corrobore les conclusions de travaux précédent, mais qui va aussi rajouter des nuances, notamment avec l'introduction d'un classement de qualité de la source, ainsi que son implication géopolitique. Dans le fichier `extsearch/news/base/search/relevance.cpp`, on voit l'exceptionnalisme de certains pays:

NEWS RELEVANCE

Il y a donc bien des traces de gestion algorithmique de documents journalistiques, avec des dynamiques de réarrangement, de *boost*, de facteurs qui permettent de déterminer la pertinence d'un résultat. Ce qui transparait néanmoins, c'est que ce qu'on va appeler un algorithme, ce sont surtout des structures de fonctions, de données, et de pondérations d'apprentissage machine qui sont difficilement démêlables, même si on peut remarquer les priorités des personnes qui ont écrit ces codes sources.

autres modules

Je vais m'arrêter là, mais je voudrais aussi signaler d'autres directions de recherche, pour ceux et celles qui seraient intéressé.es, notamment **Alice**, et la collection et gestion de données d'un assistant personnel (qu'est-ce qui est entendu, comment, et qu'est-ce qui est transféré et sauvegardé par Yandex), ainsi que **Eats**, et la gestion algorithmique des travailleurs des plateformes (comment les travailleurs sont-ils définis et gérés dans le code).

Des questionnements - 1min

Pour conclure, j'ai plusieurs questions autour de ce projet dont on pourrait discuter ensuite:

- Quelle est la valeur de ces documents? Comment est-ce qu'elle vient compléter des recherches et méthodologies existantes? Quid du fétichisme du code source?
- Au niveau méthodologique, je me demande aussi par où on commence? comment on cherche? Est-ce qu'il y a des spécificités liées à une analyse critique du code source, où est-ce qu'il suffit de le traiter simplement comme un document en language naturel? Est-ce que vous avez des recommandations de logiciels pour explorer des grandes bases de code?
- Comment est-ce qu'on prend en compte l'absence de documents? Notamment les pondérations des réseaux de neurones, ou bien les fichiers de données en production?
- Avant de mener une analyse critique, est-ce qu'il faut d'abord mener une analyse tout court? Comment est-ce qu'on conjugue interprétation fonctionnelle avec interprétation critique?