Анализ и улучшение автоматической генерации патчей с помощью машинного обучения

Беляев Станислав Валерьевич

руководители: Брыксин Т.А., Шпильман А.А.

СПб АУ НОЦНТ РАН

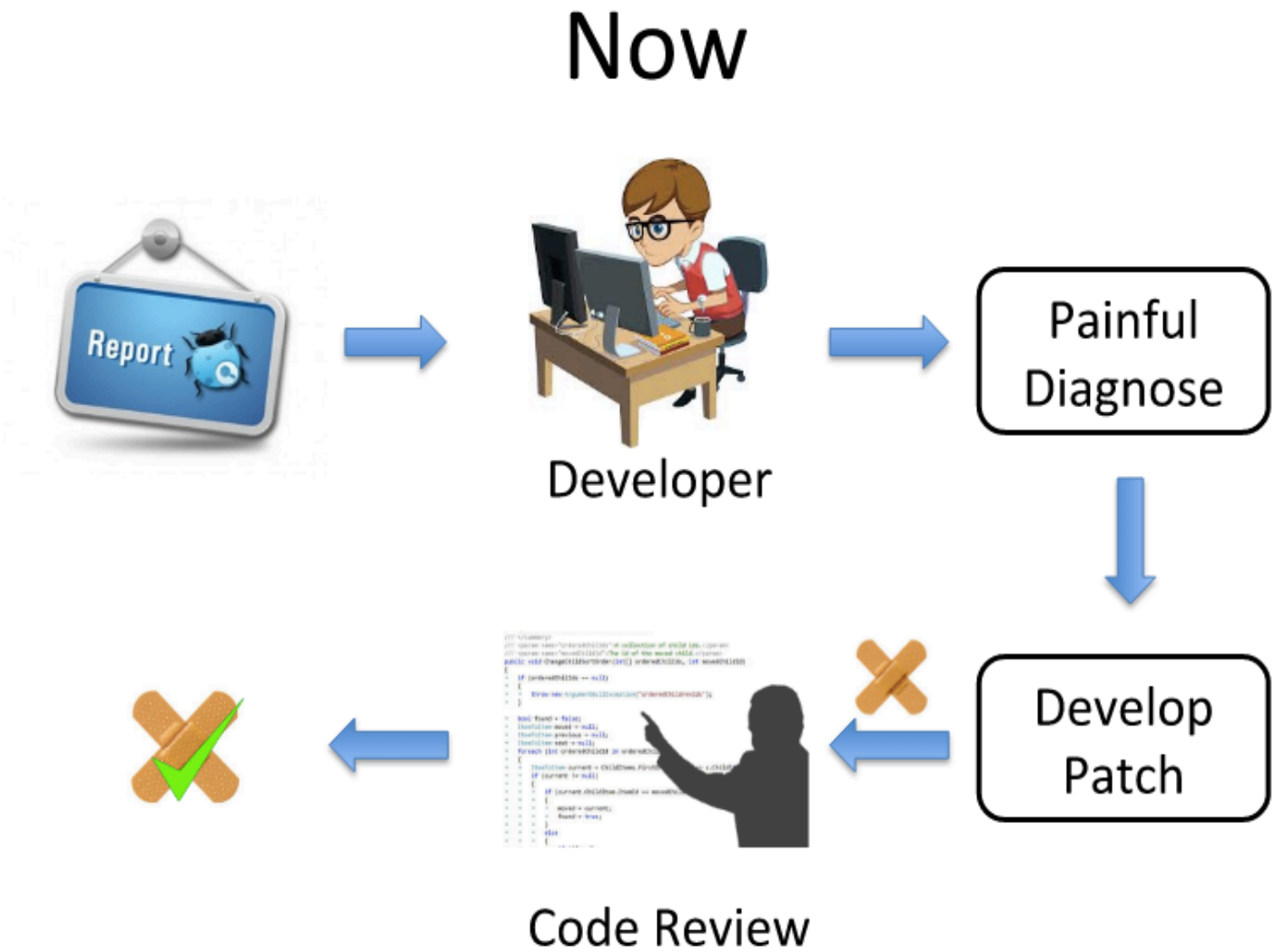
1 марта 2017 г.

Предметная область

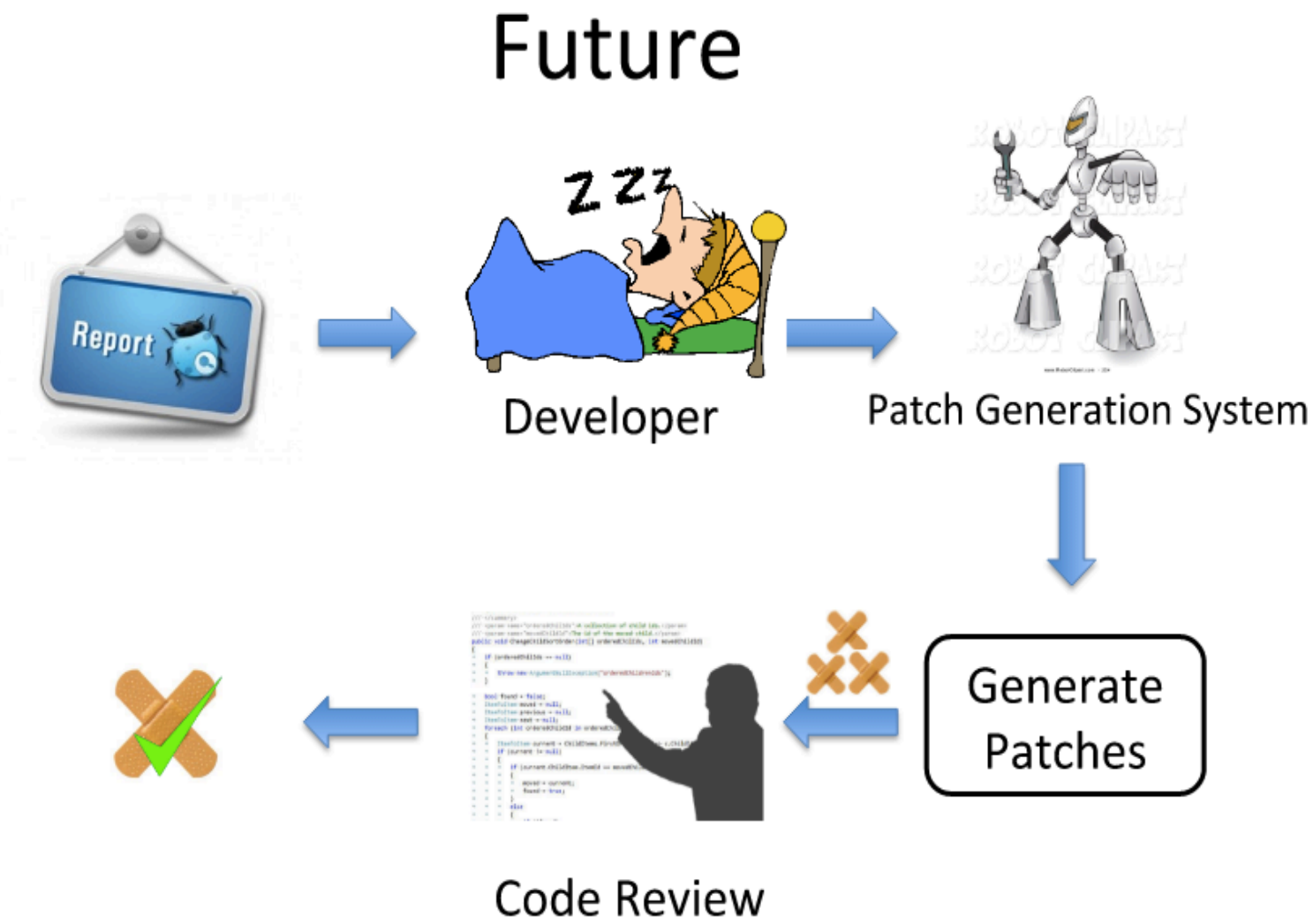
Написано много кода, а в нем много ошибок.
Актуальная задача:

- автоматически сгенерировать патчи
- валидация с помощью тестов
- проверка корректности

Предметная область



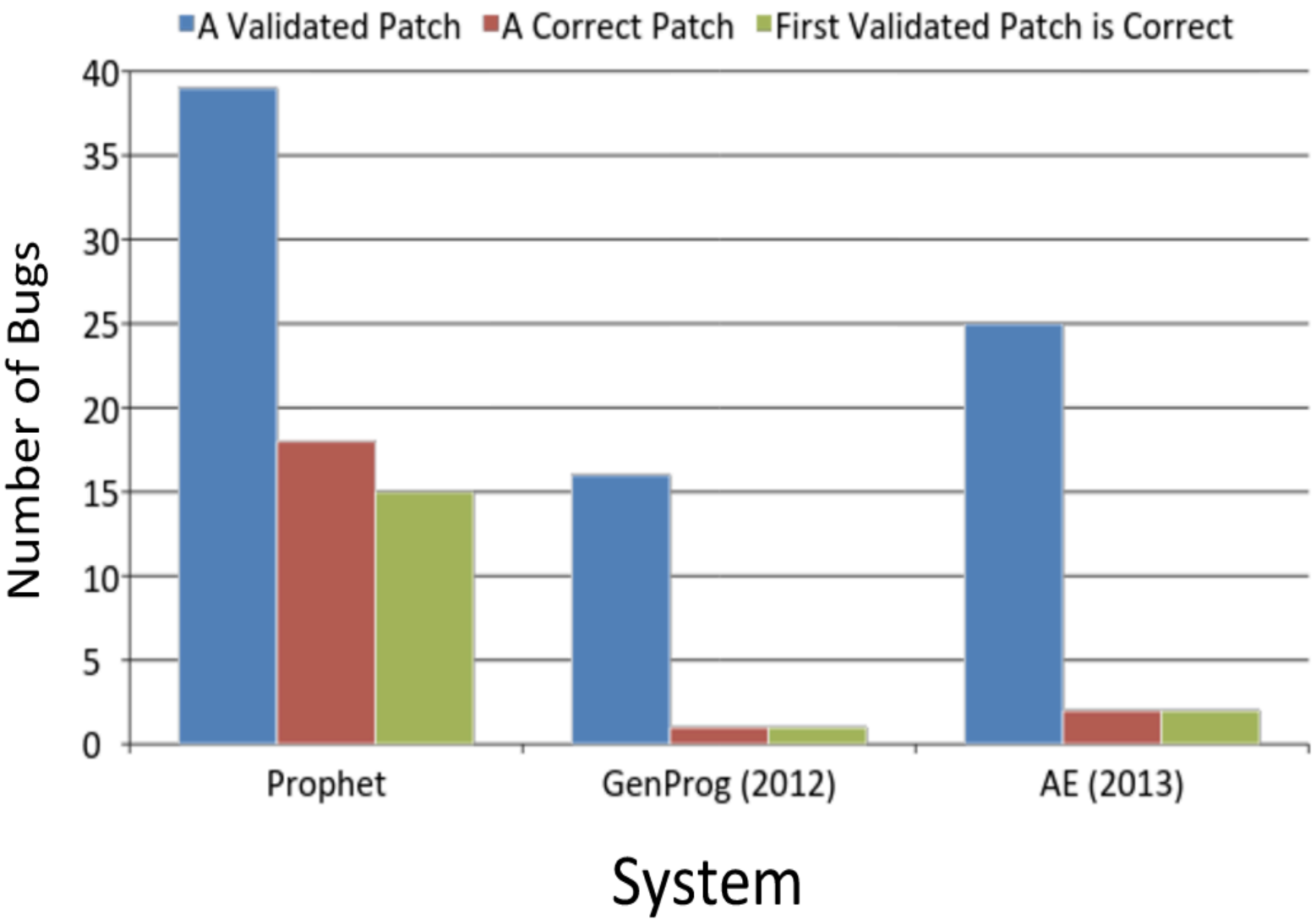
Предметная область



Предметная область

- Метод, основанный на поиске из пространства патчей (generate and validate)
  - GenProg
  - PAR
  - SPR
  - **Prophet** (State-of-the-art)
- Метод, основанный на семантическом анализе
  - Nopol
  - SimeFix, DirectFix
  - **Angelix**

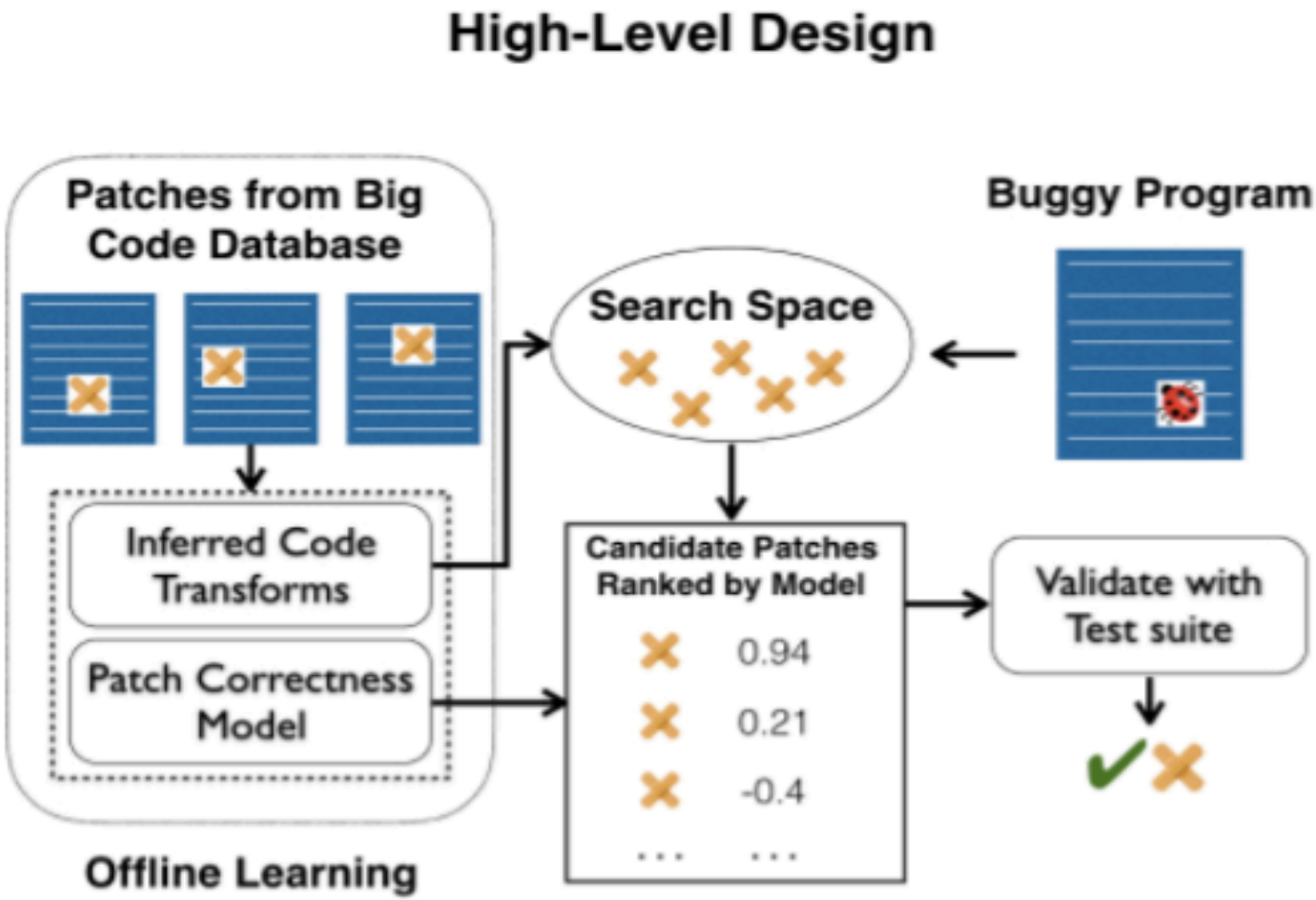
Предметная область



Prophet

Есть программа P, ненулевые наборы плохих и хороших тестов, параметр обучения:

- этап локализации: необходимо найти потенциально плохие места в программе
- этап генерации пространства патчей (generate)
- этап ранжирования патчей
  - program value features - как переменные и константы использовались до и после патча
  - modification features - какое преобразование применили и код какого типа рядом
  - при сборе фич мы абстрагируемся от деталей синтаксиса
  - на сборе фичей с успешных патчей мы обучаем вероятностную модель
- этап валидации (validate)


High-Level Design
Patches from Big Code Database
Buggy Program
Search Space
Inferred Code Transforms
Patch Correctness Model
Candidate Patches Ranked by Model
0.94
0.21
-0.4
Validate with Test suite
Offline Learning

Insert Control Flow

S ⟹ if (**E**) return **K**; S

S ⟹ if (**E**) goto **L**; S

S ⟹ if (**E**) break; S

Insert Guard

S ⟹ if (**E**) { S }

Change Condition

if (C) {…} else {…} ⟹ if (C && **E**) {…} else {…}

if (C) {…} else {…} ⟹ if (C || **E**) {…} else {…}

Replace

S ⟹ S[e1/e2]

S ⟹ S[c1/c2]

S ⟹ S[f(e1, …, en)/g(e1,…,en)]

Copy

S ⟹ Q[e1/e2]; S

S ⟹ Q[c1/c2]; S

S ⟹ Q[f(e1, …, en)/g(e1,…,en)]; S

Initialize

S ⟹ memset(&e, 0, sizeof(e)); S

Пример (PHP bug #54283):

```
char *isostr = NULL;
int isostr_len = 0;
parse(&isostr, &isostr_len);
if (isostr_len) {...}
```

Корректный патч (семантически эквивалентен реальному патчу):

```
if (isostr_len || (isostr != 0)) {...}
```

Просто некорректный патч:

```
exit(0);
if (isostr_len) {...}
```

Некорректный патч, проходящий тесты (!!!):

```
if (isostr_len && (ht == 1)) {...}
```

Инженерная задача

Дано:

- исходники с ошибками и дубликатами файлов
- работающий, но медленный AWS-container (community) из Северной Вирджинии по ssh
  - ограничения по памяти
  - ограничения по времени
  - дальше - платно (но и так 10$ сняли)
- статья с описанием работы
- почта авторов
- набор проектов-бенчмарков с зависимостями от старых версий (используется еще с GenProg)

Инженерная задача

Задача:

- контейнер с окружением для работы Prophet’а
  - удобно мигрировать на сервер
  - удобно пересобирать проект
- работающая тестирующая система
  - подтвердить результаты из статьи
  - работать разумное время (а не $12n$ часов)
- удобная совместная работа над проектом

Инженерная задача

Результаты:

- https://hub.docker.com/r/stasbel/prophet/ с подробной инструкцией (в README на гит-хабе)
  - https://github.com/StasBel/prophet-test и https://github.com/StasBel/prophet-src
  - скрипт для начального тестирования, запуска с разделением этапов и всякой удобной мелочи
  - скрипт для воспроизведения бенчмарков
  - скрипт для сборки в локальном контейнере
  - скрипт для быстрого запуска на своем проекте
- тестирующая система, запускающая несколько проектов с ограничением по времени и считающая всякие метрики у результата
  - результаты статьи подтвердились цифра в цифру
  - их можно подтвердить самостоятельно, откатившись до первых коммитов
- подправлены исходники
  - общение по почте с разработчиками
  - правка кода и удаление дубликатов

Dive into source code

Что успел сделать?

- добавил некоторые комментарии к коду, чтобы легче было разобраться
- пофиксить ошибку с неправильным подсчетом числа патчей
- добавить информацию в логи для лучшего анализа работы
- Улучшил этап локализации

Этап локализации

- $\texttt{loc} = (file, str, col)$ - выражение
- профайлером можно собрать:
  - neg - число отрицательных тестов проходящих через loc
  - pos - число положительных тестов проходящих через loc
  - $\{b_t^+\}$ - массив расстояний (в тиках) до завершения программы на положительных тестах
  - $\{b_t^-\}$ - массив расстояний (в тиках) до завершения программы на отрицательных тестах
- было: $f(\texttt{loc}) = 10^6\ \text{neg} - \text{pos}$
- стало: $f(\texttt{loc}) =$

$$\left[\frac{\text{neg}}{\text{negs}} - \frac{\text{pos}}{\text{poss}} + \exp\left(-\frac{\sum_{b\in\{b_t^-\}}\left[b\cdot\frac{1}{e^{8b}}\right]}{8\sum_{b\in\{b_t^-\}}\left[\frac{1}{e^{8b}}\right]}\right) - \exp\left(-\frac{\sum_{b\in\{b_t^+\}}\left[b\cdot\frac{1}{e^{8b}}\right]}{8\sum_{b\in\{b_t^+\}}\left[\frac{1}{e^{8b}}\right]}\right)\right]$$

- локализация влияет на этап ранжирования ($\beta \approx 0.02$)

$$P(\texttt{patch}) = \left[(1-\beta)^{\texttt{rank(patch)}}\right] \times \dots$$

- результаты:

Test	Patches	AverageRank	FirstFixRatio	FirstSchemaRatio	Time
before	1	1007.33333	0.018492	0.00038	1hour
after	2	262	0.00816	0.00037	1hour

Идеи

- распараллеливание
  - один поток - один патч, точно будет лучше
  - один поток - один тест на одном патче, скорее всего нет, т.к. overhead на переключение
- большая часть тестов скорее всего не взаимодействует со схемой
  - много тестов нужны для предотвращения регрессии
  - можно построить граф вызовов и обрубать ветки
- генерация пространства патчей
  - пространство используется еще с SPR
  - можно собирать частые изменения ast програмно
- скрестить с SMT-solver’ами
  - Angelix

Результаты

- Изучил предметную область
- Инженерная задача: docker-контейнер
- Улучшил алгоритм локализации и другие мелкие фиксы
- Использовал:
  - Python, Bash
  - C++, C
  - Docker
- Ссылки:
  - https://github.com/StasBel/prophet-src
  - https://github.com/StasBel/prophet-test
  - https://hub.docker.com/r/stasbel/prophet/