**Техлид. Аналитик. Ruby on Rails.**

Сергей Виноградов

ДР: 13.02.1988

**Мой YouTube канал с примерами моей работы:** <https://www.youtube.com/channel/UCslx2pd0dAYNv8Aygb9PhIQ>

**Образование:**

2005-2009-Национальный Университет львовская Политехника, Компьютерная Лингвистика (искусственный интеллект с уклоном на создание программ-переводчиков).

**Самообразование:**

2012 - Software as a Service от Barkeley (показывающий аспекты управления веб-платформами и Ruby on Rails).

2019-2020 Онлайн курсы математики от Massachusetts Institute of Technology и Arizona State University (с нуля до высшей математики, включая Линейную Алгебру, Многомерный Анализ и т.п.)

2010-по сегодняшнее время – постоянно изучаю лимиты возможностей Ментальных Карт и аналитики с их использованием.

**О себе:**

Беру активный проект или бизнес-план, и составляю полную техническую архитектуру!

Вбиваю в ментальную карту всё от повторяющихся технических процессов и связей между ними до возможных повторяющихся проблем и т.п.  и на основании этого составляю исчерпывающее техническое задание. Таким образом, мы можем начать с сырой идеи, запущенного проекта или просто бизнес-плана и закончить полной окончательной версией проекта, полностью автоматизированной, с решёнными стратегическими проблемами и т.п.

Могу выбрать оборудование, и реализовать технические решения (программы и т.п.).

Образцы моей авторской ментальной карты можете скачать здесь: <https://sergevinogradoff.me/>

Примеры её использования на реальных проектах – на моём YouTube канале.

**Навыки:**

## MindManager

Прежде чем что-то программировать, нужно составить план проекта (будь то активного проекта или бизнес-плана). Я могу с использованием ментальных карт взять любой (в том числе сырой) проект, спланировать его повторяющиеся процессы, предвидеть возможные повторяющиеся проблемы, составить архитектуру и конечное техническое задание. Такой метод оберегает от многих стратегических ошибок ещё на первых этапах разработки. Это самый мощный наш инструмент. Смотрите на нашем YouTube канале способы его применения.

Майндменеджером я увлекаюсь очень давно и знаю многие грани потенциала этого мощного инструмента. Я считаю это самой сильной своей стороной.

## Ruby on Rails

Чудесный язык для серверной обработки данных, позволяющий быстро и надёжно писать сложные веб-проекты. Когда-то я перешёл на Ruby on Rails ради синтаксиса и понял, что могу сделать то что не потянет команда из 5 хороших специалистов на PHP. Я этот язык использовал ещё с 2012го - задолго до того, как он стал мейнстримом.

Несколько лет я в одиночку вёл бэк энд нескольких довольно больших социальных сетей с серверной обработкой данных, финансовыми транзакциями и т.п.

## ****PostgreSQL****

Сверхнадёжная база данных, созданная университетом Беркли ещё в 1980х, и с тех пор активно развивается, и все эти годы активно используется не только для всех сложнейших проектов типа соц сетей с сверхвысокими нагрузками, но и устанавливается на железо, которое потом запускают спутники в космос для хранения научных данных.

Я читал книжку по Postgres 2 раза, но всё равно со временем это забывается, так как не является моим главным направлением, по этому я всё равно всё что связано с процедурами и сложным SQL делаю держа под рукой документацию.

## Angular.js

Библиотека созданная Google для обработки данных на стороне клиента. Позволяет перенести сложные, но не особо важные операции на вычислительную мощность людей, посещающих сайт. На сегодняшний день является самой развитой для этих задач.

Я комфортно себя чувствую в инструментах типа Karma, для меня целиком понятен принцип компонентности, но я всё таки иногда могу что-то погуглить потому что я больше по аналитике.

## Pinegrow

Инструмент позволяющий быстро и качественно верстать сложные сайты, создавать веб дизайны, подключать кастомные джаваскриптовые библиотеки и т.д.

## Twitter Bootstrap Studio

Инструмент позволяющий ещё быстрее создавать качественные сайты-визитки и вёрстки на основе уже готовых темплейтов на основе Twitter Bootstrap. Идеален для первых стадий разработки.

## Лучшие Практики

Полное тестовое покрытие (с использованием Test Driven Development, Behaviour Driven Development и библиотек типа RSpec, Karma и т.п.).

Грамотное применение практик Объектно-Ориентированного Программирования (подхода, когда за каждую сущность проекта отвечает один объект).

Однострочность методов (принцип "один метод - одна строчка") - главный показатель того, что код хорошо протестирован, продуман, и не написан затычками на колене.

Выбор лучших библиотек (которые по стилю кода соответствуют вышеупомянутым требованиям наших проектов). Хороший рецепт начинается с отличных ингредиентов.

## Уровень безопасности CIS Level 1 или CIS Level 2

В мире веб-безопасности есть такая вещь как Center Internet Security (CIS). Это база знаний всех уязвимостей всех слоёв системы (Linux, базы данных и т.п.). Есть несколько уровней защиты от этих уязвимостей:

* CIS Level 1 (отключить все возможные уязвимости, отключение которых не влияет на работу - например обычный сервер).
* CIS Level 2 (отключить все возможные функции операционных систем, которые могут быть уязвимостью, даже если их отключение может привести к трудностям работы системы).

Обычно хватает Level 1. Level 2 это для банков и прочих учреждений с финансовыми транзакциями и хранением финансовых данных, где есть повышенное требования к безопасности, и где проводятся ежегодные аудиты безопасности.

Руки пока не дошли в этом от и до разобраться. Я понимаю, что для этого нужен Chef. Видел, как выглядит книга настроек линукса под эти уровни безопасности. Но пока не применял. В мои планы входит в ближайшее время в этом разобраться.

## Оптимизация под высокие нагрузки

Обычно код на классическом Ruby on Rails + Postgres + Angular оптимизирован и очень легко справляется с нагрузками в 10,000 запросов в минуту (чего запросто хватает для 99% проектов). Для более высоких нагрузок есть ряд приёмов, которые, не требуя длительных вложений увеличат производительность в разы. А именно:

* Оптимизация SQL (выполнение нескольких задач за один запрос, перенесение вычислений с уровня приложения на уровень базы данных, подключение баз данных типа Redis для обработки данных)
* Плагины небольшие на С/С++ (вообще Ruby on Rails и так всего лишь оболочка для С, но всё-таки иногда некоторые задачи выносят в чистый С для максимизации скорости).

По мере надобности могу сделать стресс-тест чтоб определить ботлнеки системы, если они не были решены ещё на этапе построения плана.

## Информационная Архитектура

Могу составить техническое задание всего проекта в целом (будь то активно работающего или просто бизнес-плана), написать техническое решение под конкретные задачи и архитектуру проекта.

## Мета-программирование

Для выполнения проектов часто использую редактор VIM. Он позволяет видеть какие шаги нужно выполнить для выполнения задачи, и написать частично скрипты чтоб код генерировали скрипты, а не люди. Также сегодня я активно записываю повторяющиеся процессы моей разработки, чтобы потом выносить их в проекты, которые это автоматизируют и облегчают жизнь. Метапрограммирование это ключ к тому, чтоб делать быстрее и не стопориться на мелочах.

**Техническое обслуживание компьютеров, ноутбуков, сетей**

Могу быстро установить операционные системы на один компьютер или сеть компьютеров.

Установить программы и настройки на сеть десятков-сотен компьютеров с использованием программы быстрого клонирования типа Chef и Team Viewer.

## Могу настроить сеть - Обжать кабеля, настроить роутеры и DHCP сервер, дать компьютерам права доступа друг к другу, настроить сетевые принтеры и прочее.

## Чистка компьютеров

## Могу разобрать аккуратно машину (или сеть машин - т.е. компьютеров, ноутбуков и т.п.)

## Грамотно почистить специальными насадками пылесоса и выдуванием сжатым воздухом. Отмыть изопропиловым спиртом и сверхмягкими кисточками. Что дальше не отмылось - отмыть специальными дистиллированными растворами в ультразвуке.

## Будет как новый!

## Замена запчастей

## Заменить матрицу, сгоревшие или залитые микросхемы, повреждённые части корпуса, залипшие сгоревшие кулеры и прочее.

## Консалтинг по апгрейду компьютеров

## Диск на сегодняшний день советуем ставить диск SSD Apacer AS350 (на 240Гб - самый лучший из бюджетных) - либо если бюджет позволяет - то Samsung EVO последних моделей на 500Гб (показатели скорости у них примерно одинаковые, но у Samsung гарантия 5 лет и возможностей восстановления и защиты данных больше).

## Оперативки в машину на сегодняшний день советуем минимум 6-8Гб.

## Процессор обычно советуем начиная от Ryzen 3, но если надо другие модели то сравниваем их по показателям бенчмарков типа [UserBenchmark.com](https://www.userbenchmark.com/) и [CpuBenchmark.net](https://www.cpubenchmark.net/) и по показателям системных мониторов типа [CPU-Z](https://www.cpuid.com/softwares/cpu-z.html).

Видеокарту выбираю под конкретную задачу, обычно с использованием всё тех же [UserBenchmark.com](https://www.userbenchmark.com/) и [VideoCardBenchmark.net](https://www.cpubenchmark.net/)  и системных мониторов типа [GPU-Z](https://www.techpowerup.com/gpuz/).

Блоки питания обычно только EVGA.

## Помощь при покупке б/у компьютеров

## Если нужно сэкономить бюджет и выбрать компьютер с ОЛХ и пойти на встречу с продавцом или заказать что-то в Интернете, мы могу:

## Посмотреть нет ли подозрительных зазоров. Посмотреть нету ли криво посаженных портов и кнопок (и других признаков неверной разборки, которую продавец скрывает). Пропустить компьютер по чеклисту (включение, дует ли кулер, реагируют ли все кнопки на нажатие, работают ли все порты, нет ли засветов на матрице, работает ли правая-левая колонки и т.п.).

## Всё для того, чтобы купить хороший компьютер!

## Настройка прочей периферии

* Камер Видеонаблюдения
* Систем бекапов
* Проекторов
* Мониторов
* Записывающих Устройств
* Прочей аппаратуры

## Немного о моём опыте работы.

Работал 1 год как фуллтайм руби-бекенд программист (мидл+) на политической соц сети Ruck.us которая делалась под выборы в США 2014го года (с нуля всё от обработки данных и страниц юзеров до финансовых транзакций и т.п.). С момента запуска эта соц сеть собрала около миллиона долларов на донейтах, которые люди отправляли политическим кандидатам, которые регистрировались на этой соц сети.

До этого тянул свои проекты пару лет (один по аналитике просмотров объявлений на Craigslist, и второй по логистике айфонов). На которых, собственно, и понял важность бестпрактисов (однострочных методов, тестового покрытия, и, собственно, даже языка Руби, который благодаря отсутствию дурацких ограничений синтаксиса позволял мне в одиночку тянуть проекты, которые по-хорошему делали бы человек 5 на популярном на тот момент РНР. Плюс, Rails уже тогда шла с Postgres, который работал куда стабильней на нагрузках аналитики объявлений, которые достигали у меня 10,000 запросов в час).

На тот момент я делал всё в проектах сам, от кода до бизнес-аналитики. Я мог по 3 раза в неделю переезжать в новый город для бизнес-договоров и стратегических встреч. И пробовал как-то ментальными картами решать бизнес-аналитику. Но каждый раз, когда я пробовал в своих проектах разобраться картой, я понимал, что строю "идеальную надёжную лестницу не к той стенке". И в какой-то момент я подумал, что может стоит попробовать отложить делать бизнес-аналитику для кого-то, а просто попробовать писать код для людей, которые понимают куда надо идти.

Я ещё тогда яростно превращал любые проявления «простыни кода» в «код однострочных методов по стайлгайдам».

Например, с такого («один метод = простыня ифов»):

def process(*data*)  
 *bytes* = StringIO.new(String(*data*)).each\_byte  
 *bytes*.each do |*byte*|  
 case *byte* when REPORT\_VERSION  
 @major\_version = *bytes*.next  
 @minor\_version = *bytes*.next  
  
 emit('report\_version')  
  
 when ANALOG\_MESSAGE\_RANGE  
 *least\_significant\_byte* = *bytes*.next  
 *most\_significant\_byte* = *bytes*.next  
  
 *value* = *least\_significant\_byte* | (*most\_significant\_byte* << 7)  
 *pin* = *byte* & 0x0F  
  
 if *analog\_pin* = analog\_pins[*pin*]  
 pins[*analog\_pin*].value = *value* emit('analog-read', *pin*, *value*)  
 emit("analog-read-#{*pin*}", *value*)  
 end  
  
 when DIGITAL\_MESSAGE\_RANGE  
 *port* = *byte* & 0x0F  
 *first\_bitmask* = *bytes*.next  
 *second\_bitmask* = *bytes*.next  
 *port\_value* = *first\_bitmask* | (*second\_bitmask* << 7)  
  
 8.times do |*i*|  
 *pin\_number* = 8 \* *port* + *i* if *pin* = pins[*pin\_number*] and *pin*.mode == INPUT  
 *value* = (*port\_value* >> (*i* & 0x07)) & 0x01  
 *pin*.value = *value* emit('digital-read', *pin\_number*, *value*)  
 emit("digital-read-#{*pin\_number*}", *value*)  
 end  
 end  
  
 when START\_SYSEX  
 *current\_buffer* = [*byte*]  
 begin  
 *current\_buffer*.push(*bytes*.next)  
 end until *current\_buffer*.last == END\_SYSEX  
  
 *command* = *current\_buffer*[1]  
  
 case *command* when CAPABILITY\_RESPONSE  
 *supported\_modes* = 0  
 *n* = 0  
  
 *current\_buffer*.slice(2, *current\_buffer*.length - 3).each do |*byte*|  
 if *byte* == 127  
 *modes* = []  
 *# the pin modes* [INPUT, OUTPUT, ANALOG, PWM, SERVO].each do |*mode*|  
 *modes*.push(*mode*) unless (*supported\_modes* & (1 << *mode*)).zero?  
 end  
  
 @pins.push(Pin.new(*modes*, OUTPUT, 0))  
  
 *supported\_modes* = 0  
 *n* = 0  
 next  
 end  
  
 *supported\_modes* |= (1 << *byte*) if *n*.zero?  
  
 *n* ^= 1  
 end  
  
 emit('capability\_query')  
  
 when ANALOG\_MAPPING\_RESPONSE  
 *pin\_index* = 0  
  
 *current\_buffer*.slice(2, *current\_buffer*.length - 3).each do |*byte*|  
  
 @pins[*pin\_index*].analog\_channel = *byte* @analog\_pins.push(*pin\_index*) unless *byte* == 127  
  
 *pin\_index* += 1  
 end  
  
 emit('analog\_mapping\_query')  
  
 when PIN\_STATE\_RESPONSE  
 *pin* = pins[*current\_buffer*[2]]  
 *pin*.mode = *current\_buffer*[3]  
 *pin*.value = *current\_buffer*[4]  
  
 *pin*.value |= (*current\_buffer*[5] << 7) if *current\_buffer*.size > 6  
  
 *pin*.value |= (*current\_buffer*[6] << 14) if *current\_buffer*.size > 7  
  
 when FIRMWARE\_QUERY  
 @firmware\_name = *current\_buffer*.slice(4, *current\_buffer*.length - 5).reject { |*b*| *b*.zero? }.map(&:chr).join  
 emit('firmware\_query')  
  
 else  
 puts 'bad byte'  
 end  
 end  
 end  
rescue StopIteration  
 *# do nadda*end

На такой («один метод = одна строчка»):

scope :visible\_to\_user, -> (user) { where(is\_public: true).or(where(user: user)) }  
scope :without\_source, -> (record\_a) { where.not(id: record\_a.id) }  
scope :of\_type, -> (type) { where(record\_type: type) }  
  
scope :accessible\_record\_bs, -> (user, record\_a) { visible\_to\_user(user).without\_source(record\_a) }  
scope :accessible\_record\_bs\_by\_type, -> (user, record\_a, type) { accessible\_record\_bs(user, record\_a).of\_type(type) }  
scope :accessible\_record\_bs\_by\_subtype, -> (user, record\_a, type, subtype) { accessible\_record\_bs\_by\_type(user, record\_a, subtype).joins(:connections\_as\_target).where(connections: { record\_a: (accessible\_record\_bs\_by\_type(user, record\_a, type) ) })}  
  
scope :all\_roots, -> { where.not(id: Connection.pluck(:record\_b\_id)) }  
scope :siblings, -> (user, record\_a) { where(id: Record.children(Record.parents(record\_a)) ).without\_source(record\_a) }  
scope :children, -> (parent\_ids) { Connection.where(record\_a\_id: parent\_ids).pluck(:record\_b\_id) }  
scope :parents, -> (record\_a) { Connection.where(record\_b\_id: record\_a.id).pluck(:record\_a\_id) }  
  
scope :all\_parents\_of\_record, -> (record) { where(id: ActiveRecord::Base.connection.execute(all\_parent\_ids(record)).pluck('id')).without\_source(record) }  
scope :all\_tree\_records\_of\_record, -> (record) { where(id: ActiveRecord::Base.connection.execute(all\_tree\_record\_ids(record)).pluck('id')) }  
  
scope :deep\_siblings, -> (record) { all\_tree\_records\_of\_record(record).where.not(id: Record.all\_parents\_of\_record(record)).without\_source(record) }  
scope :root, -> (record) { all\_tree\_records\_of\_record(record).where.not(id: all\_tree\_records\_of\_record(record).joins(:connections\_as\_target).joins(:connections\_as\_target)).without\_source(record) }  
scope :parents\_specific\_type, -> (record, \_type) { all\_parents\_of\_record(record).where(record\_type: \_type).without\_source(record) }  
scope :closest\_of\_type, -> (record, \_type) { all\_parents\_of\_record(record).where(record\_type: \_type).order(created\_at: :desc).limit(1) }  
  
scope :without\_potential\_cycles, -> (record) { where.not(id: ActiveRecord::Base.connection.execute(all\_parent\_ids(record)).pluck('id')).without\_source(record) }  
scope :all\_without\_cycles, -> (record) { without\_potential\_cycles(record).without\_source(record) }  
scope :all\_roots\_without\_cycles, -> (record) { all\_roots.without\_potential\_cycles(record).without\_source(record) }  
  
scope :all\_children\_of\_record, -> (record) { where(id: ActiveRecord::Base.connection.execute(all\_child\_ids(record)).pluck('id')).without\_source(record) }  
scope :last\_children\_of\_record, -> (record) { all\_children\_of\_record(record).where.not(id: Record.joins(:connections\_as\_source).where(id: Record.all\_children\_of\_record(record).pluck(:id))) }  
  
after\_create :recalculate\_cached\_tree\_counters  
  
def self.only\_solved  
 all.select do |record|  
 Record.all\_solved\_tree\_records\_of\_record(record).pluck(:id).include? record.id  
 end  
end  
  
def self.only\_unsolved  
 all.select do |record|  
 Record.all\_unsolved\_tree\_records\_of\_record(record).pluck(:id).include? record.id  
 end  
end  
  
def self.all\_unsolved\_tree\_records\_of\_record(record)  
 Record.where(id: ActiveRecord::Base.connection.execute(all\_unsolved\_tree\_record\_ids(record)).pluck('id'))  
end  
  
def self.all\_solved\_tree\_records\_of\_record(record)  
 Record.where(id: ActiveRecord::Base.connection.execute(all\_solved\_tree\_record\_ids(record)).pluck('id'))  
end

Этот принцип однострочности для меня открыл ещё в 2012м курс Software As a Service от Berkeley.

Я ещё тогда мог спокойно с гуглом постепенно написать рекурсивный SQL для решения ботлнеков.

Вот пример такого SQL (решавшего кучу N+1 проблем в одном из моих проектов по рекурсивному анализу связей между объектами, весь этот анализ был разбросан в приложении и приложении висло, а здесь всё выполняет один длинный SQL запрос)

def self.all\_solved\_tree\_record\_ids(record)  
 <<-SQL  
 WITH RECURSIVE solution\_connection\_type AS (#{ ConnectionType.where(name: 'Is Solved By...').limit(1).to\_sql }),  
 all\_tree\_nodes(id, path) AS (  
 SELECT id, ARRAY[id]  
 FROM records  
 WHERE id = #{record.id}  
 UNION  
 SELECT records.id, path || records.id  
 FROM all\_tree\_nodes  
 JOIN connections ON (connections.record\_b\_id = all\_tree\_nodes.id OR connections.record\_a\_id = all\_tree\_nodes.id)  
 JOIN records ON (records.id = connections.record\_a\_id OR records.id = connections.record\_b\_id)  
 WHERE NOT records.id = ANY(path)  
 ),  
 solved\_records\_ids(record\_a\_id) AS (SELECT record\_a\_id FROM connections WHERE connection\_type\_id = (SELECT id FROM solution\_connection\_type) ),  
 solved\_record\_ids\_in\_this\_tree(id) AS (SELECT id FROM all\_tree\_nodes INNER JOIN solved\_records\_ids ON all\_tree\_nodes.id = solved\_records\_ids.record\_a\_id),  
 solutions\_ids(id) AS ( SELECT record\_b\_id FROM connections WHERE connection\_type\_id = (SELECT id FROM solution\_connection\_type) ),  
 solved\_nodes\_in\_tree(id, path) AS (  
 SELECT id, ARRAY[id]  
 FROM records  
 WHERE id IN (  
 SELECT id  
 FROM solved\_record\_ids\_in\_this\_tree  
 )  
 UNION  
 SELECT \* FROM (  
 WITH solved\_nodes\_in\_tree\_inner AS (  
 SELECT \* FROM solved\_nodes\_in\_tree  
 )  
 SELECT parents.id, path || parents.id  
 FROM solved\_nodes\_in\_tree\_inner solved\_nodes\_in\_tree  
  
 JOIN connections ON (connections.record\_b\_id = solved\_nodes\_in\_tree.id)  
 AND (  
 TRUE IN (  
 SELECT destructive  
 FROM connection\_types  
 WHERE connections.connection\_type\_id = connection\_types.id  
 )  
 )  
 AND (  
 SELECT BOOL\_AND(is\_solved) AND (  
 (  
 FALSE NOT IN (  
 SELECT ct.destructive  
 FROM connections co  
 JOIN connection\_types ct  
 ON co.connection\_type\_id = ct.id  
 WHERE co.record\_a\_id = connections.record\_a\_id  
 )  
 )  
 OR  
 (  
 connections.record\_a\_id IN (  
 SELECT record\_a\_id  
 FROM solved\_records\_ids  
 )  
 )  
 )  
 FROM (  
 SELECT (  
 EXISTS (  
 SELECT \*  
 FROM solved\_nodes\_in\_tree\_inner  
 WHERE solved\_nodes\_in\_tree\_inner.id = children.id  
 )  
 ) is\_solved  
 FROM records children  
 WHERE children.id IN (  
 SELECT record\_b\_id  
 FROM connections c  
 WHERE c.record\_a\_id = connections.record\_a\_id  
 )  
 ) q  
 )  
 JOIN records parents ON (parents.id = connections.record\_a\_id)  
 WHERE (NOT parents.id = ANY(path))  
 ) t  
 ),  
 unsolved\_nodes\_in\_tree(id) AS (SELECT \* FROM all\_tree\_nodes WHERE all\_tree\_nodes.id NOT IN (SELECT id FROM solved\_nodes\_in\_tree) )  
 SELECT id FROM solved\_nodes\_in\_tree ORDER BY path  
 SQL  
end

За 1 год фуллтайм программирования соцсети на фирме по аутсорсингу я узнал, что без мета-программирования я не потяну бесконечный поток задач так чтоб хватало жизни на что-то ещё. Поэтому я перешёл на график "пол дня писать проект, а пол дня мета-программировать (например записывать какие действия повторяются, а потом смотреть по своим записям, какие действия и задачи повторяются больше всего, и потом писать Bash-скрипты, Руби-скрипты, SED-скрипты с Regex которые генерируют код. Таким методом я создавал генераторы бест практисов ещё за долго до того, как появились аналоги типа Рубокоп.

Так появился мой проект Caperoma (<https://github.com/zoopyserg/caperoma>) который частично решал проблемы, которые сегодня решает Рубокоп, а частично автоматизировал другие мелочи помимо автоматизации соблюдения стайлгайдов (а именно полностью управлял названием веток гитхаба, стартом-стопом таймеров, задач в пивотале, джире и т.п.). с помощью его я могу сэкономить до 15ти минут на каждой задаче (т.е. я выполнял сотни заданий каждые из которых я делал за 2-3 минуты, а ближайшие по зарплате мидлы в проекте делали за 15-20 минут, в результате чего я сэкономил кучу денег на разработке).

И на тот момент у меня встала дилема. Потому что так как я пол дня мета-программировал, а пол дня писал проекты, за которые мне платят - я начал писать в разы быстрее чем лучшие мидлы на фирме, всё сдавал в сроки, но в Джиру сабмитал только пол дня рабочего дня (так как я не мог засабмитить в Джиру что я что-то там метапрограммирую). Тот факт что у меня по Джире было 50% ставки, скомбинированный с тем фактом, что на момент 2013-2014го года пока люди ещё не знали насколько мощный язык Руби - ставка мидл+ рубиста была 800-1000$, я встал в дурацкую ситуацию, что работаю за дурных 400-500$ (так как сабмичу в лучшем случае 30-50% ставки) работая день и ночь над проектами с десятками тысяч юзеров, которые загребают сотни тысяч долларов на моём коде на донейтах на бекенде который я построил с нуля. В то время как у меня еле хватает на дом, и не хватает времени на семью и вообще ни на что.

В процессе работы над проектом Caperoma я понял, что мне катастрофически не хватает понимания аналитической математики и геометрии (так как по сути одним из шагов моего "генератора кода", в который много чего упёрлось, была цель "тестирования приложения искусственным интеллектом" - "прочитать слова о проблеме, и составить алгоритм чтоб воспроизвести проблему в браузере - грубо говоря кликнуть на ссылку если клиент говорит что она не работает"). Я понял, что не имеет смысла программировать на фирмах, пока не продвинусь в этом направлении хоть немного. Поэтому с 2014 по 2015й год я проектов не писал, а пытался это дело как-то решить теми знаниями, которые были. У меня это не получилось (и всё моё образование в сфере Искусственного Интеллекта во Львовском Политехе оказалось до лампочки) поэтому я начал просто фрилансить простые проекты, которые не требуют такого сильного бэкэнда, и ещё где-то год-два пробовал как-то усовершенствовать свой алгоритм "дневника повторений" (рекурсивного анализа говорящего какие задачи в проекте нужно автоматизировать в первую очередь). А когда понял, что даже этого мало - отложил всё и начал учить математику. С нуля на английском. И два года учился онлайн по всяким курсам от MIT, от Arizona State University, составлял ментальные карты математических теорем, пока не доучил всё что мог на английском с нуля до Многомерного Анализа, Линейной Алгебры и Дифференциальных Уравнений. Параллельно ради з/п я чинил компы подавая объявления в ОЛХ, немного автоматизировал бизнес-процессы клиентам своим кодом на каких-то фирмах. Подтянул джаваскрипт (где-то год писал фронт енд своих проектов на всяких Vue.js, Angular и т.п.). И только сегодня в 2021м начал пробовать снова лезть в Искусственный Интеллект (распознавание образов, дип лёрнинг и т.п.). Только теперь увидел насколько наивно было пробовать это решить не понимая математику тысяч измерений, градиентов функций и прочего. Очень надеюсь, что когда-то соберусь с силами закончить свой проект Caperoma. Так же за эти годы я сильно подтянул свои навыки аналитики до какого-то сверхэффективного максимума (способность создавать архитектуру проекта в ментальной карте информационных процессов и связей между ними), которого у меня не было даже когда я себя считал "богом майндмаппинга и анализа" а всех вокруг - "людьми, которые не знают, куда идут". Начал снимать видео на ютюбе в которых я делюсь опытом майндмаппинга, архитектуры и стратегии с другими людьми. Начал составлять для всех желающих проектов ментальные карты повторяющихся информационных процессов и связей между ними, и по этим картам автоматизировать бизнесы (не только руби-кодом, но и кодом на ещё нескольких языках - .Net, Python, железом и т.п. - постепенно заливаю это на свой YouTube канал). И вот теперь снова пробую сесть делать чьи-то проекты официально на фулл тайме, так как и так давно уже это делаю то для одного то для другого клиента.

На данный момент я веду несколько проектов (каждый из них парт-таймом):

<https://industrialovenautomation.xyz/>

<http://phasemeditationtrainer.xyz/>

И веду несколько своих сайтов-визиток:

<https://windowssetup.xyz/>

<https://webdevelopers.studio/>

<https://automations.monster/>

<https://sergevinogradoff.me/>

<https://vganalytics.live/>

И стараюсь, чтоб дошли руки и ресурсы задеплоить ещё и другие мои более важные для меня проекты чисто по аналитике.

Собираю деньги на то, чтоб собрать вычислительную мощность достаточную чтоб их открыть.

Готов параллельно поработать ещё над чужими проектами (так как польза от меня есть, и я и так уже тяну несколько активных проектов и в принципе меня на это всё хватает).

Задумался подать резюме на Руби, так как вроде как наконец люди начали платить за Руби столько сколько он реально стоит.