**Техлід. Аналітик. Ruby on Rails.**

**Сергій Виноградов**

**ДН: 13.02.1988**

**Мій YouTube канал з прикладами моєї роботи:**

<https://www.youtube.com/channel/UCslx2pd0dAYNv8Aygb9PhIQ>

**Освіта:**

2005-2009-Національний Університет львівська Політехніка, Комп'ютерна Лінгвістика (штучний інтелект із ухилом на створення програм-перекладачів).

**Самоосвіта:**

2012 - Software as a Service від Barkeley (що показує аспекти управління веб-платформами та Ruby on Rails).

2019-2020 Онлайн курси математики від Massachusetts Institute of Technology та Arizona State University (з нуля до вищої математики, включаючи Лінійну Алгебру, Багатомірний Аналіз тощо)

2010-нині – постійно вивчаю ліміти можливостей Ментальних Карт та аналітики з їх використанням.

**Про себе:**

Беру активний проект чи бізнес-план, і складаю повну технічну архітектуру!

Вбиваю в ментальну карту все від технічних процесів, що повторюються, і зв'язків між ними до можливих повторюваних проблем і т.п. і на підставі цього становлю вичерпне технічне завдання. Таким чином, ми можемо почати з сирої ідеї, запущеного проекту або просто бізнес-плану та закінчити повною остаточною версією проекту, повністю автоматизованою, із вирішеними стратегічними проблемами тощо.

Можу вибрати обладнання та реалізувати технічні рішення (програми тощо).

Зразки моєї авторської ментальної карти можете завантажити тут: <http://sergevinogradoff.me/>

Приклади її використання на реальних проектах – на моєму каналі YouTube.

**Навики:**

## MindManager

## Перш ніж щось програмувати, потрібно скласти план проекту (чи активного проекту чи бізнес-плану). Я можу з використанням ментальних карт взяти будь-який (у тому числі сирий) проект, спланувати його повторювані процеси, передбачати можливі проблеми, що повторюються, скласти архітектуру і кінцеве технічне завдання. Такий метод оберігає від багатьох стратегічних помилок ще перших етапах розробки. Це найпотужніший наш інструмент. Дивіться на нашому каналі YouTube способи його застосування.

## Майндменеджером я захоплююся дуже давно і знаю багато меж потенціалу цього потужного інструменту. Я вважаю це найсильнішою своєю стороною.

## Ruby on Rails

Чудова мова для серверної обробки даних, що дозволяє швидко та надійно писати складні веб-проекти. Колись я перейшов на Ruby on Rails заради синтаксису і зрозумів, що можу зробити те, що не потягне команда з 5 хороших фахівців на PHP. Я цю мову використав ще з 2012-го - задовго до того, як він став мейнстримом.

Кілька років я поодинці вів бек енд кількох досить великих соціальних мереж із серверною обробкою даних, фінансовими транзакціями тощо.

## ****PostgreSQL****

Наднадійна база даних, створена університетом Берклі ще в 1980-х, і з тих пір активно розвивається, і всі ці роки активно використовується не тільки для всіх найскладніших проектів типу соціальних мереж з надвисокими навантаженнями, але і встановлюється на залізо, яке потім запускають супутники в космос. зберігання наукових даних.

Я читав книгу по Postgres 2 рази, але все одно з часом це забувається, так як не є моїм головним напрямком, тому я все одно пов'язано з процедурами і складним SQL роблю тримаючи під рукою документацію.

## Angular.js

Бібліотека створена Google для обробки даних на стороні клієнта. Дозволяє перенести складні, але не особливо важливі операції на обчислювальну потужність людей, які відвідують сайт. На сьогоднішній день є найрозвиненішою для цих завдань.

Я комфортно почуваюся в інструментах типу Karma, для мене цілком зрозумілий принцип компонентності, але я все-таки іноді можу щось погуглити тому що я більше по аналітиці.

## Pinegrow

Інструмент дозволяє швидко і якісно верстати складні сайти, створювати веб-дизайни, підключати кастомні джаваскриптові бібліотеки і т.д.

## Twitter Bootstrap Studio

Інструмент, що дозволяє ще швидше створювати якісні сайти-візитки та верстки на основі вже готових темплейтів на основі Twitter Bootstrap. Ідеальний для перших стадій розробки.

**Найкращі практики**

Повне тестове покриття (з використанням Test Driven Development, Behaviour Driven Development та бібліотек типу RSpec, Karma тощо).

Грамотне застосування практик Об'єктно-орієнтованого програмування (підходу, коли за кожну сутність проекту відповідає один об'єкт).

Однорядковість методів (принцип "один метод - один рядок") - головний показник того, що код добре протестований, продуманий, і не написаний затичками на коліні.

Вибір найкращих бібліотек (які за стилем коду відповідають вищезгаданим вимогам наших проектів). Хороший рецепт починається із відмінних інгредієнтів.

**Рівень безпеки CIS Level 1 або CIS Level 2**

У світі веб-безпеки є така річ, як Center Internet Security (CIS). Це база знань всіх уразливостей всіх верств системи (Linux, бази даних тощо). Є кілька рівнів захисту від цих уразливостей:

* CIS Level 1 (вимкнути всі можливі вразливості, відключення яких не впливає на роботу, наприклад звичайний сервер).
* CIS Level 2 (вимкнути всі можливі функції операційних систем, які можуть бути вразливими, навіть якщо їх відключення може призвести до труднощів роботи системи).

Зазвичай вистачає Level 1. Level 2 це для банків та інших установ з фінансовими транзакціями та зберіганням фінансових даних, де є підвищені вимоги до безпеки, та де проводяться щорічні аудити безпеки.

Руки поки що не дійшли в цьому від і до розібратися. Я розумію, що для цього потрібний Chef. Бачив, як виглядає книга налаштувань лінукса під ці рівні безпеки. Але поки що не застосовував. У мої плани входить найближчим часом в цьому розібратися.

**Оптимізація під високі навантаження**

Зазвичай код на класичному Ruby on Rails+Postgres+Angular оптимізований і дуже легко справляється з навантаженнями в 10,000 запитів на хвилину (чого запросто вистачає для 99% проектів). Для більш високих навантажень є низка прийомів, які, не вимагаючи тривалих вкладень, збільшать продуктивність у рази. А саме:

* Оптимізація SQL (виконання кількох завдань за один запит, перенесення обчислень з рівня програми на рівень бази даних, підключення баз даних типу Redis для обробки даних)
* Плагіни невеликі на С/С++ (взагалі Ruby on Rails і так лише оболонка для С, але все-таки іноді деякі завдання виносять в чистий С для максимізації швидкості).

При необхідності можу зробити стрес-тест щоб визначити ботлнеки системи, якщо вони були вирішені ще етапі побудови плану.

**Інформаційна архітектура**

Можу скласти технічне завдання всього проекту в цілому (будь то активно працюючого або просто бізнес-плану), написати технічне рішення під конкретні завдання та архітектуру проекту.

**Мета-програмування**

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**

## Для виконання проектів часто використовую редактор VIM. Він дозволяє бачити які кроки необхідно виконати до виконання завдання, і написати частково скрипти щоб код генерували скрипти, а чи не люди. Також сьогодні я активно записую повторювані процеси моєї розробки, щоб потім виносити їх у проекти, які це автоматизують та полегшують життя. Метапрограмування це ключ до того, щоб робити швидше і не стопоритися на дрібницях.

## Чищення комп'ютерів

## Можу розібрати акуратно машину (або мережу машин – тобто комп'ютерів, ноутбуків тощо)

## Грамотно почистити спеціальними насадками пилососа та видуванням стисненим повітрям.

## Відмити ізопропіловим спиртом та надм'якими пензликами.

## Що далі не відмилося – відмити спеціальними дистильованими розчинами в ультразвуку.

## Буде як новий!

## Заміна запчастин

## Замінити матрицю, згорілі або залиті мікросхеми, пошкоджені частини корпусу, кулери, що залипли згорілі, та інше.

## Консалтинг з апгрейду комп'ютерів

## Диск на сьогоднішній день радимо ставити диск SSD Apacer AS350 (на 240Гб – найкращий з бюджетних) – або якщо бюджет дозволяє – то Samsung EVO останніх моделей на 500Гб (показники швидкості у них приблизно однакові, але у Samsung гарантія 5 років та можливостей відновлення та захисту даних більше).

## Оперативки в машину на сьогоднішній день радимо мінімум 6-8Гб.

## Процесор зазвичай радимо починаючи від Ryzen 3, але якщо треба інші моделі, то порівнюємо їх за показниками бенчмарків типу [UserBenchmark.com](https://www.userbenchmark.com/) і [CpuBenchmark.net](https://www.cpubenchmark.net/) та за показниками системних моніторів типу [CPU-Z](https://www.cpuid.com/softwares/cpu-z.html).

Відеокарту вибираю під конкретне завдання, зазвичай з використанням все тих же [UserBenchmark.com](https://www.userbenchmark.com/) і [VideoCardBenchmark.net](https://www.cpubenchmark.net/)  та системних моніторів типу [GPU-Z](https://www.techpowerup.com/gpuz/).

## Блоки живлення зазвичай лише EVGA.

## Допомога при купівлі б/у комп'ютерів

## Якщо потрібно заощадити бюджет і вибрати комп'ютер із ОЛХ та піти на зустріч із продавцем або замовити щось в Інтернеті, ми можу:

## Подивитися чи немає підозрілих проміжків.

## Подивитися чи немає криво посаджених портів і кнопок (та інших ознак неправильного розбирання, яке продавець приховує).

## Пропустити комп'ютер за чеклістом (включення, чи дме кулер, чи реагують всі кнопки на натискання, чи працюють всі порти, чи немає засвітів на матриці, чи працює права-ліва колонки і т.п.).

## Все для того, щоб купити гарний комп'ютер!

## Налаштування іншої периферії

## Камер Відеоспостереження

## Систем бекапів

## Проекторів

## Моніторів

## Записуючих пристроїв

## Інші апаратури

**Трохи про мій досвід роботи.**

Працював 1 рік як фуллтайм рубі-бекенд програміст (мідл+) на політичній соціальній мережі Ruck.us яка робилася під вибори в США 2014-го року (з нуля все від обробки даних і сторінок користувачів до фінансових транзакцій тощо). З моменту запуску ця соціальна мережа зібрала близько мільйона доларів на донейтах, які люди відправляли політичним кандидатам, які реєструвалися на цій соціальній мережі.

До цього тягнув свої проекти кілька років (один з аналітики переглядів оголошень на Craigslist, і другий за логістикою айфонів). На яких, власне, і зрозумів важливість беспрактісів (однорядкових методів, тестового покриття, і, власне, навіть мови Рубі, який завдяки відсутності безглуздих обмежень синтаксису дозволяв мені поодинці тягнути проекти, які по-доброму робили б людина 5 на популярній на той момент РНР Плюс, Rails вже тоді йшла з Postgres, який працював куди стабільнішим на навантаженнях аналітики оголошень, які досягали у мене 10,000 запитів на годину).

На той момент я робив все в проектах сам, від коду до бізнес-аналітики. Я міг по 3 рази на тиждень переїжджати до нового міста для бізнес-договорів та стратегічних зустрічей. І намагався якось ментальними картами вирішувати бізнес-аналітику. Але щоразу, коли я пробував у своїх проектах розібратися картою, я розумів, що будую "ідеальні надійні сходи не до тієї стінки". І в якийсь момент я подумав, що може варто спробувати відкласти бізнес-аналітику для когось, а просто спробувати писати код для людей, які розуміють куди треба йти.

Я ще тоді люто перетворював будь-які прояви «простирадла коду» на «код однорядкових методів по стайлгайдах».

Наприклад, з такого («один метод = простирадло іфів»):

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 такий («один метод = один рядок»):

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

Цей принцип однорядковості для мене відкрив ще в 2012 курс Software As a Service від Berkeley.

Я ще тоді міг спокійно з гуглом поступово написати рекурсивний SQL для вирішення ботлнеків.

Ось приклад такого SQL (вирішував купу N+1 проблем в одному з моїх проектів з рекурсивного аналізу зв'язків між об'єктами, весь цей аналіз був розкиданий у додатку та додатку висло, а тут все виконує один довгий SQL запит)

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 рік фуллтайм програмування соцмережі на фірмі з аутсорсингу я дізнався, що без мета-програмування я не потягну нескінченний потік завдань так, щоб вистачало життя на щось ще. Тому я перейшов на графік "півдня писати проект, а пів дня мета-програмувати (наприклад записувати які дії повторюються, а потім дивитися за своїми записами, які дії та завдання повторюються найбільше, і потім писати Bash-скрипти, Рубі-скрипти, SED-скрипти з Regex які генерують код.Таким методом я створював генератори беспрактісів ще за довго до того, як з'явилися аналоги типу Рубокоп.

Так з'явився мій проект Caperoma (<https://github.com/zoopyserg/caperoma>) який частково вирішував проблеми, які сьогодні вирішує Рубокоп, а частково автоматизував інші дрібниці, крім автоматизації дотримання стайлгайдів (а саме повністю керував назвою гілок гітхабу, стартом-стопом таймерів, завдань у півоталі, джирі тощо). за допомогою його я можу заощадити до 15 хвилин на кожному завданні (тобто я виконував сотні завдань кожні з яких я робив за 2-3 хвилини, а найближчі по зарплаті мідли в проекті робили за 15-20 хвилин, в результаті чого я заощадив купу грошей на розробці).

І на той момент у мене постала проблема. Тому що я пів дня мета-програмував, а пів дня писав проекти, за які мені платять - я почав писати в рази швидше за кращі мідли на фірмі, все здавав у строки, але в Джиру саботив тільки пів дня робочого дня (так тому що я не міг знайти в Джиру що я щось там метапрограмую). Той факт, що у мене по Джирі було 50% ставки, скомбінований з тим фактом, що на момент 2013-2014 року поки люди ще не знали наскільки потужна мова Рубі - ставка мідл + рубіста була 800-1000 $, я встав у безглузду ситуацію, що працюю за поганих 400-500 $ (оскільки сабмічу в кращому разі 30-50% ставки) працюючи день і ніч над проектами з десятками тисяч користувачів, які загребають сотні тисяч доларів на моєму коді на донейтах на бекенді який я побудував з нуля. У той час як у мене ледве вистачає на хату, і не вистачає часу на сім'ю та взагалі ні на що.

У процесі роботи над проектом Caperoma я зрозумів, що мені катастрофічно не вистачає розуміння аналітичної математики та геометрії (оскільки по суті одним із кроків мого "генератора коду", в який багато чого вперлося, була мета "тестування програми штучним інтелектом" - "прочитати" слова про проблему, і скласти алгоритм щоб відтворити проблему в браузері - грубо кажучи натиснути на посилання якщо клієнт каже що вона не працює "). Я зрозумів, що немає сенсу програмувати на фірмах, поки не просунуся в цьому напрямку хоч трохи. Тому з 2014 по 2015 рік я проектів не писав, а намагався цю справу якось вирішити тими знаннями, які були. У мене це не вийшло (і вся моя освіта у сфері Штучного Інтелекту у Львівському Політеху виявилася до лампочки) тому я почав просто фрілансити прості проекти, які не вимагають такого сильного бекенду, і ще десь рік-два пробував якось удосконалити свій алгоритм "щоденника повторень" (рекурсивного аналізу, який говорить, які завдання в проекті потрібно автоматизувати в першу чергу). А коли зрозумів, що навіть цього мало – відклав усе і почав вивчати математику. З нуля англійською. І два роки навчався онлайн за будь-якими курсами від MIT, від Arizona State University, становив ментальні карти математичних теорем, поки не вивчив все що міг англійською з нуля до Багатомірного Аналізу, Лінійної Алгебри та Диференціальних Рівнянь. Паралельно заради з/п я лагодив комп'ютери подаючи оголошення в ОЛХ, трохи автоматизував бізнес-процеси клієнтам своїм кодом на якихось фірмах. Підтягнув джаваскрипт (десь рік писав фронт енд своїх проектів на всяких Vue.js, Angular тощо). І тільки сьогодні у 2021-му почав пробувати знову лізти в Штучний Інтелект (розпізнавання образів, диплернінг тощо). Тільки тепер побачив, наскільки наївно було пробувати це вирішити не розуміючи математику тисяч вимірів, градієнтів функцій та іншого. Дуже сподіваюся, що колись зберусь із силами закінчити свій проект Caperoma. Так само за ці роки я сильно підтягнув свої навички аналітики до якогось надефективного максимуму (здатність створювати архітектуру проекту в ментальній карті інформаційних процесів та зв'язків між ними), якого у мене не було навіть коли я себе вважав "богом майндмапінгу та аналізу". всіх довкола - "людьми, які не знають, куди йдуть". Почав знімати відео на ютюбі, в яких я ділюся досвідом майндмапінгу, архітектури та стратегії з іншими людьми. Почав складати для всіх бажаючих проектів ментальні карти повторюваних інформаційних процесів і зв'язків між ними, і по цих картах автоматизувати бізнеси (не тільки рубі-кодом, а й кодом ще кількома мовами - .Net, Python, залізом тощо - поступово заливаю це на свій канал YouTube. І ось тепер знову пробую сісти робити чиїсь проекти офіційно на фул таймі, тому що і так давно вже це роблю то для одного, то для іншого клієнта.

На даний момент я веду кілька проектів (кожен із них парт-таймом):

<https://industrialovenautomation.xyz/>

<http://phasemeditationtrainer.xyz/>

І веду кілька своїх сайтів-візиток:

<https://windowssetup.xyz/>

<https://webdevelopers.studio/>

<https://automations.monster/>

<https://sergevinogradoff.me/>

<https://vganalytics.live/>

І намагаюся, щоб дійшли руки та ресурси задеплоїти ще й інші мої важливіші для мене проекти суто за аналітикою.

Збираю гроші на те, щоб зібрати обчислювальну потужність, достатню щоб їх відкрити.

Готовий паралельно попрацювати ще над чужими проектами (оскільки користь від мене є, і я і так уже тягну кілька активних проектів і в принципі мене на це все вистачає).

Задумався подати резюме на Рубі, тому що начебто люди почали платити за Рубі стільки, скільки він реально коштує.