**Tech Lead. Analyst. Ruby on Rails.**

Serge Vinogradoff

BD: 13.02.1988

**My YouTube channel with examples of my work:**

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

**Education:**

2005-2009 - National University Lviv Polytechnic, Computational Linguistics (artificial intelligence with a focus on creating translation programs).

**Self-education:**

2012 - Software as a Service by Barkeley (showing aspects of web platform management and Ruby on Rails).

2019-2020 Online mathematics courses from Massachusetts Institute of Technology and Arizona State University (from zero to advanced mathematics, including Linear Algebra, Multivariate Analysis, etc.)

2010-present time - I am constantly studying the limits of the possibilities of Mental Maps and analytics with their use.

**About myself:**

I take an active project or business plan and draw up a complete technical architecture!

I drive everything into the mental map from repetitive technical processes and connections between them to possible recurring problems, etc. and on the basis of this I draw up an exhaustive technical task. Thus, we can start with a raw idea, a launched project or just a business plan and end with a full final version of the project, fully automated, with resolved strategic problems, etc.

I can choose equipment and implement technical solutions (programs, etc.).

Samples of my author's mental map can be downloaded here: <https://sergevinogradoff.me/>

Examples of its use on real projects - on my YouTube channel.

## Skills:

## MindManager

## Before programming anything, you need to draw up a project plan (whether it be an active project or a business plan). Using mental maps, I can take any (including a raw) project, plan its recurring processes, anticipate possible recurring problems, draw up an architecture and a final technical assignment. This method protects against many strategic mistakes even in the early stages of development. This is our most powerful tool. Check out our YouTube channel for how to use it.

## I have been fond of mind-manager for a very long time and I know many facets of the potential of this powerful tool. I think this is my greatest strength.

## Ruby on Rails

A wonderful server-side data processing language that allows you to quickly and reliably write complex web projects. Some time ago I switched to Ruby on Rails for the sake of syntax and realized that I could do something that a team of 5 good PHP specialists couldn't handle. I have been using this language since 2012 - long before it became mainstream.

For several years I have been single-handedly running the back-end of several fairly large social networks with server-side data processing, financial transactions, etc.

## ****PostgreSQL****

An ultra-reliable database created by the University of Berkeley back in the 1980s, and since then has been actively developing, and all these years has been actively used not only for all the most complex projects such as social networks with ultra-high loads, but also installed on hardware, which is then launched by satellites into space for storage of scientific data.

I read a book on Postgres 2 times, but all the same, over time it is forgotten, since it is not my main focus, so I still do everything related to procedures and complex SQL keeping documentation at hand.

## Angular.js

A library created by Google for client-side data processing. Allows you to transfer complex, but not particularly important operations to the computing power of people visiting the site. Today it is the most developed for these tasks.

I feel comfortable in tools like Karma, for me the principle of components is completely clear, but I still can sometimes google something because I am more of an analyst.

## Pinegrow

A tool that allows you to quickly and efficiently make up complex sites, create web designs, connect custom javascript libraries, etc.

## Twitter Bootstrap Studio

A tool that allows you to even faster create high-quality business card sites and layouts based on ready-made templates based on Twitter Bootstrap. Ideal for the early stages of development.

## Best Practices

Full test coverage (using Test Driven Development, Behavior Driven Development and libraries like RSpec, Karma, etc.).

Competent application of the practices of Object-Oriented Programming (an approach when one object is responsible for each entity of the project).

One-line methods (the principle of "one method, one line") is the main indicator that the code is well tested, well thought out, and not written with knee gags.

Selection of the best libraries (which in terms of code style correspond to the aforementioned requirements of our projects). A good recipe starts with great ingredients.

**Security Level CIS Level 1 or CIS Level 2**

In the world of web security, there is such a thing as Center Internet Security (CIS). This is a knowledge base of all vulnerabilities of all layers of the system (Linux, databases, etc.). There are several layers of protection against these vulnerabilities:

* CIS Level 1 (disable all possible vulnerabilities, the disabling of which does not affect the work - for example, a regular server).
* CIS Level 2 (disable all possible functions of operating systems that may be a vulnerability, even if their disabling may lead to difficulties in the system).

Usually, Level 1 is enough. Level 2 is for banks and other institutions with financial transactions and storage of financial data, where there are increased security requirements, and where annual security audits are conducted.

I didn’t learn everything about it for now. I understand that Chef is needed for this. I saw what the book of Linux settings for these security levels looks like. But I haven't used it yet. It is in my plans to figure it out in the near future.

**Optimization for high loads**

Typically, the code in classic Ruby on Rails + Postgres + Angular is optimized and can handle loads of 10,000 requests per minute very easily (which is easily enough for 99% of projects). For higher loads, there are a number of tricks that, without requiring long-term investments, will increase productivity significantly. Namely:

* SQL optimization (performing several tasks in one query, transferring calculations from the application level to the database level, connecting databases like Redis for data processing)
* Small plugins in C / C ++ (in general, Ruby on Rails is just a shell for C, but still, sometimes some tasks are taken out to pure C to maximize speed).

If necessary, I can do a stress test to determine system bottlenecks, if they were not solved at the stage of building the plan.

**Information Architecture**

I can draw up a technical assignment for the entire project as a whole (whether it is actively working or just a business plan), write a technical solution for specific tasks and project architecture.

**Meta programming**

I often use the VIM editor to complete projects. It allows you to see what steps need to be performed to complete the task, and to write partial scripts so that the code is generated by scripts, not people. Also today, I actively record the repetitive processes of my development, so that later they can be taken out into projects that automate this and make life easier. Metaprogramming is the key to getting things done faster and not getting stuck with little things.

**Maintenance of computers, laptops, networks**

## I can quickly install operating systems on one computer or a network of computers.

## Install programs and settings on a network of tens to hundreds of computers using fast cloning software such as Chef and Team Viewer.

## I can set up a network - Crimp cables, set up routers and a DHCP server, give computers access rights to each other, set up network printers, and more.

## Cleaning computers

## I can neatly disassemble a machine (or a network of machines - i.e. computers, laptops, etc.)

## Clean properly with special vacuum cleaner nozzles and blowing out with compressed air.

## Wash with isopropyl alcohol and super soft brushes.

## What is not washed further - wash with special distilled solutions in ultrasound.

## It will be like new!

## Replacement of spare parts

## Replace the matrix, burnt or flooded microcircuits, damaged parts of the case, stuck burned-out coolers, etc.

## Computer upgrade consulting

## Today we recommend installing an Apacer AS350 SSD (240GB is the best of the budget) - or if the budget allows, then Samsung EVO of the latest 500GB models (their speed indicators are about the same, but Samsung has a 5-year warranty and recovery options and data protection more).

## We recommend at least 6-8GB of RAM in the car today.

## We usually recommend the processor starting from Ryzen 3, but if you need other models, then we compare them in terms of benchmarks such as [UserBenchmark.com](https://www.userbenchmark.com/) and [CpuBenchmark.net](https://www.cpubenchmark.net/) and according to the indicators of system monitors such as [CPU-Z](https://www.cpuid.com/softwares/cpu-z.html).

I choose a video card for a specific task, usually using the same [UserBenchmark.com](https://www.userbenchmark.com/) and [VideoCardBenchmark.net](https://www.cpubenchmark.net/)  and system monitors like [GPU-Z](https://www.techpowerup.com/gpuz/).

## Power supplies are usually EVGA only.

## Help with buying used computers

## If you need to save a budget and choose a computer with CFM and go to a meeting with a seller or order something online, we can:

## See if there are any suspicious gaps.

## See if there are any crooked ports and buttons (and other signs of incorrect disassembly, which the seller is hiding).

## Skip the computer according to the checklist (turning on, whether the cooler is blowing, whether all buttons react to pressing, whether all ports are working, are there any glare on the matrix, is the right-left column working, etc.).

## Everything to buy a good computer!

## Configuring other peripherals

## CCTV Cameras

## Backup systems

## Projectors

## Monitors

## Recorders

## Other equipment

**A little about my work experience.**

He worked for 1 year as a full-time ruby-backend programmer (middle +) on the political social network Ruck.us, which was made for the US elections in 2014 (from scratch everything from data processing and user pages to financial transactions, etc.). Since its launch, this social network has raised about a million dollars in donations that people sent to political candidates who registered on this social network.

Prior to that, I was dragging my projects for a couple of years (one in the analytics of ad views on Craigslist, and the second in the logistics of iPhones). On which, in fact, I understood the importance of best practices (one-line methods, test coverage, and, in fact, even the Ruby language, which, due to the absence of stupid syntax restrictions, allowed me to single-handedly pull projects that would be done in an amicable way by 5 people on the popular at that time PHP: Plus, Rails already came with Postgres, which worked much more stable on ad analytics workloads, which reached 10,000 requests per hour for me).

At that time, I did everything in projects myself, from code to business intelligence. I could move to a new city 3 times a week for business contracts and strategic meetings. And I tried to solve business analytics with mental maps. But every time I tried to figure out a map in my projects, I realized that I was building "an ideal reliable staircase to the wrong wall." And at some point I thought that maybe I should try to postpone doing business analytics for someone, but just try to write code for people who understand where to go.

Even then, I was furiously turning any manifestation of the "code sheet" into "code of one-line methods according to style guides."

For example, with this ("one method = ifov sheet"):

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

To this ("one method = one line"):

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

This one-line principle was discovered for me back in 2012 by the Software As a Service course from Berkeley.

Even then, I could easily write recursive SQL with Google to solve botnets.

Here is an example of such SQL (which solved a bunch of N + 1 problems in one of my projects on recursive analysis of relationships between objects, all this analysis was scattered in the application and the application was hanging, and here everything is performed by one long SQL query)

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

During 1 year of full-time programming of a social network at an outsourcing company, I learned that without meta-programming I cannot pull an endless stream of tasks so that there is enough life for something else. Therefore, I switched to the schedule "to write a project for half a day, and meta-programming for half a day (for example, write down which actions are repeated, and then look at your notes, which actions and tasks are repeated the most, and then write Bash scripts, Ruby scripts, SED scripts with Regex that generate code This is the method I used to create best practice generators long before there were analogues like Rubokop.

This is how my project Caperoma was born (<https://github.com/zoopyserg/caperoma>) which partially solved the problems that Rubokop solves today, and partially automated other little things in addition to automating the observance of style guides (namely, completely managing the name of the github branches, start-stop timers, tasks in the pivotal, jir, etc.). using it, I can save up to 15 minutes on each task (i.e., I performed hundreds of tasks, each of which I did in 2-3 minutes, and the middle salaries nearest in salary in the project were done in 15-20 minutes, as a result of which I saved a ton of money on development).

And at that moment I had a dilemma. Because since I was meta-programming half a day, and half a day I was writing projects for which they pay me, I started writing many times faster than the best middle at the firm, I handed everything in on time, but in Jira I submitted only half a day of a working day (so how I could not submit to Jiru that I was metaprogramming something there). The fact that I had 50% of the stake on Jira, combined with the fact that at the time of 2013-2014, while people still did not know how powerful Ruby's language was - the middle + Rubyst rate was $ 800-1000, I got into a stupid situation, that I work for a bad $ 400-500 (since submission is at best 30-50% of the rate) working day and night on projects with tens of thousands of users who rake in hundreds of thousands of dollars on my donate code on a backend that I built from scratch. While I barely have enough time to spend at home, and do not have enough time for my family and for anything else.

In the process of working on the Caperoma project, I realized that I was sorely lacking in understanding of analytical mathematics and geometry (since in fact one of the steps of my "code generator", into which a lot of things rested, was the goal of "testing the application with artificial intelligence" - "read words about the problem, and create an algorithm to reproduce the problem in the browser - roughly speaking, click on the link if the client says that it does not work "). I realized that it makes no sense to program in firms until I have made some progress in this direction. Therefore, from 2014 to 2015, I did not write projects, but tried to somehow solve this matter with the knowledge that I had. I didn't succeed (and all my education in the field of Artificial Intelligence at the Lviv Polytech turned out to be a light bulb), so I just started freelancing simple projects that do not require such a strong backend, and for about a year or two I tried to somehow improve my algorithm of the "repetition diary" (recursive analysis of the speaker which tasks in the project need to be automated in the first place). And when I realized that even this was not enough, I put everything aside and began to study mathematics. From scratch in English. And for two years he studied online in all kinds of courses from MIT, from Arizona State University, made mental maps of mathematical theorems, until he finished everything he could in English from scratch to Multidimensional Analysis, Linear Algebra and Differential Equations. At the same time, for the sake of salary, I repaired computers by submitting ads to the OLH, slightly automated business processes for clients with my code at some companies. I pulled up javascript (for about a year I wrote the front end of my projects in all sorts of Vue.js, Angular, etc.). And only today, in 2021, I began to try again to climb into Artificial Intelligence (pattern recognition, deep-running, etc.). Only now I saw how naive it was to try to solve this without understanding the mathematics of thousands of dimensions, gradients of functions and other things. I really hope that someday I will muster the strength to finish my Caperoma project. Also, over the years, I have strongly tightened my analytics skills to some super-efficient maximum (the ability to create a project architecture in a mental map of information processes and connections between them), which I did not have even when I considered myself "the god of mind mapping and analysis" all around - "people who do not know where they are going." I started shooting videos on YouTube in which I share my experience of mind mapping, architecture and strategy with other people. He began to draw up mental maps of recurring information processes and connections between them for all interested projects, and using these maps to automate businesses (not only with Ruby code, but also with code in several more languages ​​- .Net, Python, hardware, etc. - gradually upload it to my YouTube channel). And now I again try to sit down to do someone's projects officially at full time, since for so long I have been doing this for one or another client.

At the moment I am running several projects (each of them is part-time):

<https://industrialovenautomation.xyz/>

<http://phasemeditationtrainer.xyz/>

And I run several of my business card sites:

<https://windowssetup.xyz/>

<https://webdevelopers.studio/>

<https://automations.monster/>

<https://sergevinogradoff.me/>

<https://vganalytics.live/>

And I try to get my hands and resources around to deploy my other more important projects for me purely in analytics.

I collect money to collect enough computing power to open them.

I am ready to work in parallel on other people's projects (since there is some benefit from me, and I already pull several active projects and, in principle, I have enough for this).

I thought about applying for Ruby, because it seems like finally people started paying for Ruby as much as it really costs.