Sergei Shilko

www.sshilko.com

I am a senior PHP developer with 13+ years of experience in web development. Currently living in Berlin and working as a *“Head of Engineering”* - developing backend infrastructure and API’s for high-load social and language learning application.

Currently responsibilities include extending the features on the backend side of the application: environment setup, architecture, deployment, maintenance, monitoring, scaling and also management work needed to keep up the growing team size.

In my free time, i enjoy spending time with my family or working on scaling and architectual problems, reading books related to IT, project management, agile, interesting stories from industry veterans.
I stay up to date by listening to podcasts, visiting meetups and conferences, following daily tech blogs.

Interests

1. Robert Martin: Clean Code: A Handbook of Agile Software Craftsmanship
   a. Optimize decisions by delaying and obtaining full information needed (179)
   b. Prefferable selected decision, out of may - the simplest one
   c. Code is like a city, has two phases, first - construction, second - operations
   d. Building complex system from scratch is a myth, cities are build from villages, noone will approve building a city in a middle of nowhere (186)
   e. Iterative development is the key to building complex systems (186)
   f. Software architecture unlike physical building can be modified gradually, key is correct sharing of responsibility
   g. AOSD - aspect oriented programming - universal way of building modular systems (modularity.info)
   h. Law of Demeter - loose coupling of modules, close relationships circle
   i. Functions should do one thing w/o sideeffects
   j. Avoid returning NULL - NPE hell in Java
   k. Objects vs Structs. Hybrid is worse than both. Objects declare their behaviour. Structs declare their properties.
   l. DTO - Step towards objects, Structure with open properties w/o methods
   m. Bean objects - private properties, constructor with all properties & getters, setters
   n. ActiveRecord - DTO with business logic mixin (load, save, update …)
   o. Business logic inside ActiveRecord is wrong
   p. DI & Single sesponsibility principle
   q. Classes should define interfaces (i.e. mock testing)
   r. Interfaces - borderline components and responsibility handoffs
   s. Incomplete tests worse than none, give illusion of reliability
   t. TDD - dont write code before tests
   u. Scalable architecture by "Extreme Programming Explained", Kent Beck, 1999
      i. All tests pass
      ii. No duplicate code pieces
      iii. Expresses developer intention
      iv. Has ninimal required amount of code/classes
   v. The next person who is going to use your code is probably going to be you (yourself)
2. *Adrenaline Junkies and Template Zombies: Understanding Patterns of Project Behavior*
3. *Edward Yourdon Death March*
4. *Alan Kuper The Inmates Are Running the Asylum, ISBN 0-672-31649-8*
5. *Smart and Gets Things Done: Joel Spolsky`s Concise Guide to Finding the Best Technical Talent*
6. *Frederick P. Brooks, The Mythical Man-Month*
7. *E. Gamma Design Patterns. Elements of Reusable Object-Oriented Software*
8. *J. Hank Rainwater Herding Cats: A Primer for Programmers Who Lead Programmers*
9. *Allan Kelly: Xanpan*
   a. *"There are two ways of constructing a software design: one way is to make it so simple that there are obviously no deficiencies, and other ways is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult. " - C A R Hoare, 1980 Turig Award Lecture.*
   b. *"Most forms of testing average only about 30% to 35% in defect removal efficiency levels and*

seldom top 50%. Formal design and code inspections, on the other hand, often top 85% in defect removal efficiency and average about 65% …'

   c. *"Formal design and code inspections are the most effective defect removal activities in the history of software and are also very good in terms of defect prevention (Jones 2008)"*
   d. *"For the next few years companies which can adopt and master TDD can gain significant competitive advantage"*
   e. *"I believe that by 2020 programmers who do not practise TDD will not be able to find employment"*
   f. *"Quality is Free" (Crosby 1980)*
   g. *"The bottom line is that poor-quality sofware costs more to build and to maintain than high-quality software, and it can also degrade operational performance, increase user error rates, and reduce revenue by decreasing the ability handle customer transactions or attract additional clients"*
   h. *"Unless a team is actively working to improve software quality, not only will Xanpan fail, but any attempts at Agile are also likely to fail"*
   i. *"Specifically, if a team is not practicing test-driven development as they write code, they are a) probably not Agile, and b) likely to encounter problems in the near future."*
   j. *"It always takes longer than you expect, even when you take in account Hofstadter's Law" (Hofstadter 1980).*

10. *Karl Seguin: The Little Go Book*
11. *Valve: Handbook for new employees (Valve Press 2012)*
12. *Jinesh Varia: AWS Architecting for the Cloud: Best practices*
13. *MongoDB: Performance Best Practices (December 2015, MongoDB Whitepaper)*
14. *High Performance MySQL: Optimization, Backups, and Replication 3rd Edition by Baron Schwartz, Peter Zaitsev and Vatim Tkachenko, ISBN 978-1-449-31428-6*
15. *The Phoenix Project: A Novel About IT by Gene Kim, Kevin Behr, George Spafford, ISBN 978-0-9882625-0-8*
    a. *Business projects*
    b. *Internal projects*
    c. *Changes*
    d. *Unplanned work*
    e. *Constrains / optimizatons*
16. *SCRUM The Art of Doing Twice the Work in Half the TIme by Jeff Sutherland, ISBN 978-1-847-94110-7*
17. *The soul of a new machine by Tracy Kidder, 1981, ISBN 987-0-316-49170-9*
    a. *Putting a life in people's jobs*
    b. *Nothing ever happens unless you push it*
    c. *Take the lemons and make lemonade*
    d. *The only good strategy is one that no one else understands*
    e. *Everything depends on you, they say*
    f. *Not Everything Worth Doing Is Worth Doing Well*
    g. *Trust is the risk, and risk avoidance is the name of the game in business*
18. *Learning Swift building apps for macos, ios, and beyond, by Jonathon Manning O'REILLY 2017*
19. *The five dysfunctions of a team: a Leadership fable / Patrick Lencioni, 2002, ISBN 0-7879-6075-6*
20. *Crossing the chasm: marketing and selling high-tech products to mainstream customers / Geoffrey A. Moore*
    a. *Technology Adoption Lifecycle*
    b. *The whole product concept*
    c. *The elevator test (claim)*

   d. *If you dont know where you are going, you probably arent going to get there*
   e. *… fundamental principle for crossing the chasm is to target a specific niche market as your point of attack and focus all your resources on acheiving the dominant leadership position in that segment…*
   f. *If two people buy the same product for the same reason but have no way they could reference each other, they are not part of the same market*

21. *Designing Data-Intensive Applications: The big ideas behind reliable, scalable and maintainable systems - by Martin Kleppmann, O'Reilly 2017*
    a. *Reliability, Scalability, Maintainability*
    b. *Relational vs Object / JSON / Network-model, Hierarchical model (one-one, one-many, many-many)*
    c. *Query optimizer, query language: SQL, Cypher, SPARQL, Datalog*
    d. *Hash indexes, SST, LSM-T, B-Tree, OLAP / OLTP, Stars&Snowflakes, DataWarehouse / Column-Store*
    e. *Marshalling: JSON, THRIFT, PBuffers, (g)RPC, Actors, Message-Passing, Akka, Erlang*
22. The Deadline: A Novel about Project Management by Tom DeMarco
23. Solid Code: Optimizing the Software Development Life Cycle by Donis Marshall and John Bruno
24. The software development edge: essay on managing successful projects by Joe Marasco, ISBN 0-321-32131-6
25. Joel on software by Joel Spolsky, ISBN 1-59059-389-8
26. Why software sucks… adn what you can do about it by David S. Platt, ISBN 0-321-46675-6
27. Programming Collective Intelligence by Toby Segaran, ISBN 0-596-52932-5
28. 97 things every software architect should know: collective wisdom from the Experts, edited by Richard Monson-Haefel, ISBN 978-0-596-52269-8

Interesting law’s to keep note of

Goodhart's law	When a measure becomes a target, it ceases to be a good measure
Brooks' law	Adding human resources to a late software project makes it later
Conway's law	Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations
Dunbar's number	is a suggested cognitive limit to the number of people with whom one can maintain stable social relationships
Parkinson's law	work expands so as to fill the time available for its completion
Hofstadter's law	It always takes longer than you expect, even when you take into account Hofstadter's Law
Gall's Law	A complex system that works is invariably found to have evolved from a simple system that worked. A complex system designed from scratch never works and cannot be patched up to make it work. You have to start over with a working simple system
Law of triviality	Parkinson's law of triviality is C. Northcote Parkinson's 1957 argument that members of an organization give disproportionate weight to trivial issues.