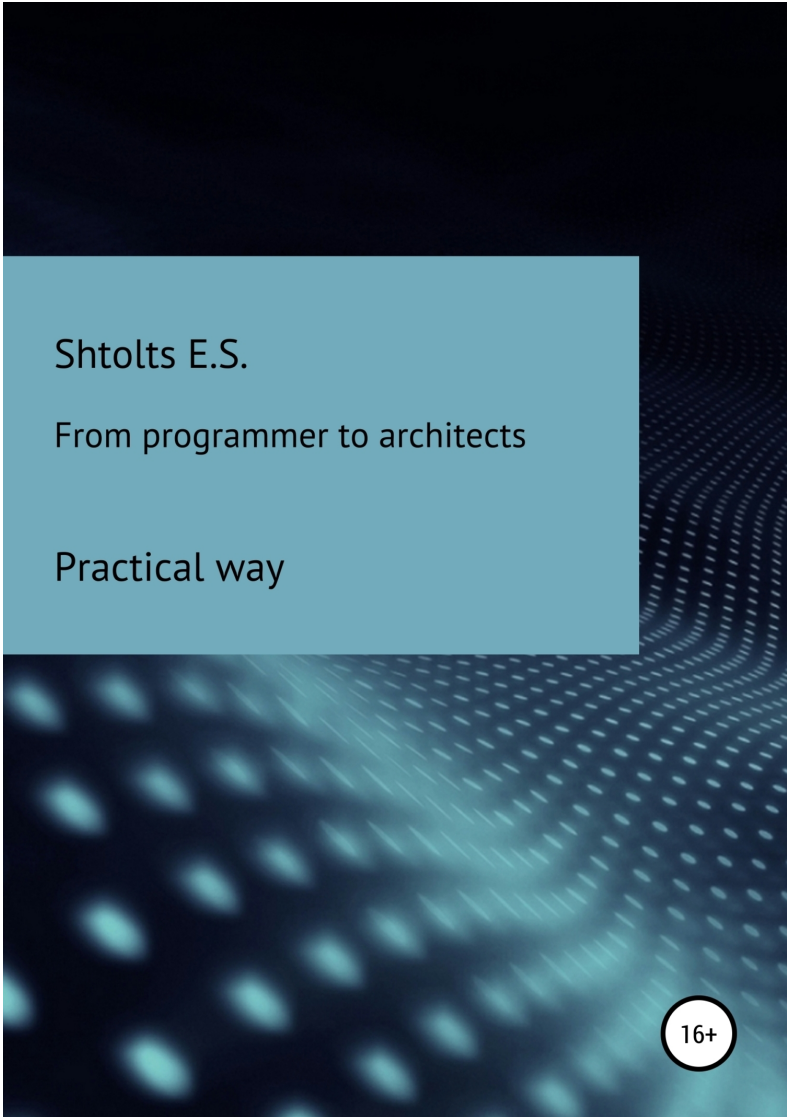
Shtolts E.S.
From programmer to architects
Practical way
16+

Eugeny Shtoltc

From programmer to architects. Practical way

http://www.litres.ru/pages/biblio_book/?art=56870818
SelfPub; 2021

Аннотация

In this book, the chief Architect of the Cloud Native Competency Architecture Department at Sberbank shares the knowledge and experience with readers gained in developing their own and evaluating other people's architectures, providing a basis for professional and career growth.

Содержание

Varieties of architects 4
Конец ознакомительного фрагмента. 20

Eugeny Shtoltc
From programmer to architects. Practical way

Varieties of architects

Architecture

ISO / IEC / IEEE 42010 defines architecture as the Basic concepts and properties of a system in the environment embodied in its elements, relationships and specific principles of its design and development. There are quite a few varieties of it, but we will single out the main ones according to the level of abstraction: application architecture (Application Architecture), software architecture (Software Architecture), application architecture Solution Architecture and business architecture (Enterprise architecture). The application architect develops the application architecture (design patterns, task allocation) and often combines his role with the role of Team-Lead and the leading developer of critical components. Software Architect does the same thing as the build architect, but works with several teams, adding the unification of the technologies

they use. Often this position is in demand in outsourcing, where there are many projects and it is possible to take the load off Team-Lead so that they communicate more with customers and the team. This position is characterized by the requirements for a vacancy in knowledge of the programming language and the main stack used on projects. In such a situation, the architect is limited in choosing technologies and hiring new employees. Since its appearance in 1959, the architect has been engaged in the decomposition of the system, the distribution of parts by developers, and was responsible for the subsequent integration of the developed components into the initially required system. Now the situation is simplified with the advent of micro-services.

The corporate architect designs the relationships between the systems using the enterprise data integration bus, and the application architect designs the systems themselves, decomposing them into applications. The boundaries between applications are determined by the boundaries of use: development, deployment, provision to the supplier. Previously, applications were also united by technological platforms and technologies, but with the advent of containerization, the situation may contain components created on different platforms, languages and stacks, enclosed in containers. Also, the formation of borders based on rolling out the application has lost its relevance due to the fact that the components (container) are rolled out and are already being tested in the environment of other components. Ideally, a group of micro services is grouped

by the function of the business and the team developing it, but often common components participate in business processes, which blurs the boundaries of applications. This specificity led to the emergence of a separate specialization – Cloud Solution Architect.

Based on the level of architecture that is supposed to be designed, it is possible to turn from an abstract question – how to become an architect – into a set of requirements necessary for solving the task: from purely technical to organizational. So a software architect can delegate all organizational activities to Team Lead and focus purely on the technical description of the program structure, and often he is a pure techie and part-time Tex-Lead, but he can not delegate the technical one. In contrast, a corporate architect may not be a technical specialist, for example, a director, conducting communication to organize communications of automated systems and meet these systems with the needs of customers. Based on this, one can guess that the question – how do they become architects – can be answered that architects before Solution Architect are evolutionary in technical branch, and corporate, either in technical branch, or managerial, including business analytics. At the same time, you can become an architect at any number of years.

Solution Architect and microservices

The introduction of microservices begins with the business,

when marketing begins to do experiments – request features in the form of MVP, then test on the market, after which either reject (which is rare) or modify it. Refinement is required both after confirmation of the assumption, and erroneous in the form of adjustments. For the maintenance service, this means rolling out a huge number of features that were developed in a hurry and can bring down the main application – the monolith. This service tries to run these changes in an isolated environment as a separate functionality, for which the development department asks for it, to develop them separately – in the form of microservices.

It is necessary to separate two levels of separation: technological and domain. Technological features in the work are the same, because the services, what its components, if it is divided into its components, are technologically launched and maintained the same way. But, unlike services, which should be minimally interconnected with other services and must provide a certain API and SLA, the components are strongly connected. The reason for the separation into components are of a technological nature. For example, an online store contains services such as the online showcase itself, payment services, storage services, and the online showcase is a service on CMS Joomla! and MySQL database management system (DBMS). Here the division of the service into its constituent parts was due to different software products written in different programming languages. At the same time, for the customer this is a single service on CMS Joomla !, performing the single function of

providing an interface for ordering to users online, provided by the hosting as a single service (it will not work separately), can work as a catalog of goods without other services (payment, order) . From a technological point of view, service components:

1. singles are not functional;

2. strongly connected, since for each request to the CMS a lot of requests are generated;

3. interaction interfaces are complex and diverse – not even the API is used here, but the SQL interaction language;

4. are strongly connected functionally through a complex technical API – for the user only the supported compatibility of some versions of CMS with other versions of the DBMS is known.

The division of the system into microservices begins with an analysis of their boundaries, an analysis of the benefits of separation and the added complexity of a distributed system. It is better to separate the world services with a combination of the need:

1. Technological need, for example, a large load that is difficult to withstand without separation, for example, scaling, another type of support (SLA);

2. For business, the allocated service is already a separate and little dependent function – then we'll consider DDD (model-driven design + ubiquitous language) approach to the implementation of microservices;

3. Need to change technology platform.

Microservice meets the following characteristics, according to M. Fowler (martinfowler.com). They can be reduced to:

1. Must be a component or service. Each microservice is a complete, fully-fledged independent service from the point of view of the developer, system administrator and user. It should have the ability to be easily replaced with another one, both in the developer code, both in the process of work (should be) and presented to others or removed in the user interface. Failure to meet the replaceability conditions at different levels leads to one service divided into parts – a distributed monolith.

2. Organization of business opportunities.

3. Products are not projects.

4. Smart endpoints and stupid connections. Not requiring complex integration with debugged services (service-oriented SOA architecture is involved in the integration of complex systems)

5. Decentralized management. This refers to orchestration, for example, Kubernetes, network management, for example, Istio, delivery management, for example, Knative.

6. Decentralized data management. Due to the self-sufficiency of the service and independence from others, it must have an independent state – the database, and for the choice of the database management system to be independent – it has its own.

7. Automated infrastructure. The process of deployment, scaling and rollbacks should be automated, which allows you to

quickly roll back automatically, fix the isolation of the service in the code.

8. There is a denial of work. To visualize failures, you can look at Jaeger and Prometheus, to localize problems, services should be isolated, present one single service, which allows isolating, limiting the harmful effects on other services in case of failure and automating the rollback.

9. Evolving design. The system builds up processes in the form of services – overgrows them, and no change in its structure is required. Neal Ford and Rebeca Parsons in Microservices as an Evolving Architecture focus on continuous improvement.

A view from the heights of a business and a business architect

Business architecture (Enterprise Architect) is the IT architecture of the entire company. She operates with abstractions and entities at the business level, these are strategies, business processes, services and the like. The systems and interconnections that support the work of a business are called the IT landscape, because it contains many systems that do not form a single whole, connected by business processes in which they participate and which are not limited to them. The business architect (Enterprise Architect) works at this level, adjusting the current landscape to the current needs of the business. Often, for traditional companies that did not develop in the high-tech

sphere in the blue ocean, it is attracted when the IT landscape has developed and difficulties arise in its development and adaptation to changing conditions, a minimally capable product (MVP) is created in technological startups. The business architect of the corporate IT landscape should solve problems with a low rate of making changes (due to the impossibility of local testing, postponing distributed changes, breaking down after rolling distributed changes) – time to market, quick adaptation to user expectations – customer experience and cost – cost. The first is solved by the architect's consistent efforts to restore order, reducing coherence and complexity, which simplifies and accelerates the process of making changes, and, like in any high-tech field, where the main cost is the man-hours of workers, reduces the cost. More and more often the requirements are not provided to the architect, he is connected at the very early stages of their formation, after his possibilities are very limited. To do this, he needs to actively participate in negotiations and meetings in order to adjust the requirements. Then form several possible solutions with different levels of complexity, from simple and fast but not effective ("solutions on the knee"), through the optimal, and to large-scale and flexible. Next, form an architectural committee on which to propose solutions for the selection and adjust the selection towards the optimal one.

Changing the existing architecture can be done in three ways: maintenance of the current, full replacement of the current, addition of the current. Replacing the current one requires a

long and lengthy study of the functionality of the current system, then clarifying the functionality that is currently in demand and conducting a search for differences between the current functionality and the expected one, after which the cost and development time are calculated. During the presentation, in most cases, a refusal to develop the system will be received, since the customer does not need a technical update of the existing functionality, but he needs new and adjustment of the old one, and especially not for the time and money that was spent on creating the current system. With a consistent improvement of the system, fundamental changes to the system cannot be achieved, since the architecturally different functionality of one part contradicts the other and changes in the architectural style are not achieved by successive improvements of the parts due to the complex nature.

Strategy Enterpris Architect differently implements different company strategies:

1. growth (scaling) strategy, when the market is free – architecture unifies and debugs processes.

2. innovation strategy (search for hypotheses by Lean Startup). Creation and delivery of features and verification as fast as possible hypotheses about the demand for these features.

3. integration strategy. An open, most scalable and versioned API needs to be developed.

4. adaptation strategy. Not a strategy, using Agile Market Tuning.

5. quick decision strategy. Successive changes.

DevOps as a component of the architect

Often in small organizations, DevOps is the only person who understands the device at all levels, and as a system administrator, the infrastructure and installed applications, and as a programmer, a developed application software part, and as an automator, business development processes. The position of an architect in large companies implies familiarity with architecture management methodologies:

1. TOGAF (The Open Group Architecture Framework) with the Archimade program;
2. Zachman Framework;
3. DoDAF;
4. ARIS (Architecture of Integrated Information Systems) with ARIS Express.

For DevOps, the general concepts of the product creation process and its implementation are important, so let's take a general one from the popular ones. If a company has an implementation specialist, usually either Tex-Lead or Team-Lead, who sees in detail how his application will be arranged and created by his team, then he can play the role of Application Application. This is a common practice and a necessary condition, since the application is highly interconnected, and you need a person who knows in detail how it works and how

changes in one part of it will affect the others. If such an examination cannot be found, the application must be divided into parts and fix the connection interfaces. To divide the project into parts, it is necessary to divide the team itself, developing it, into parts in accordance with Conway's law ("Organizations designing systems are limited to a design that copies the communication structure in this organization"). With such an organization, when the application is still one, but divided into parts, Tex-Leid and Team-Lead, having an understanding in each part and can embrace the implementation of the entire application, can get expertise from the narrower members of their teams when their expertise is not enough . If we are talking about Software Architect or Technology Architect, then now unification takes this role, respectively, in the form of containers and virtual machines. If we are talking about an application system, then it is necessary to operate with a higher level of design, such as Information Architect (System Architect and Solution Architect), in which it is impossible to combine design and implementation, since it is necessary to design a group of systems that will be developed by different teams. If the relationship is simple, and this is solved by coordinating the leads of these departments and the architect is not required. An architect is needed when the whole system is being developed and it is necessary to replace the component in the current system with a component incompatible with the current system, when it is necessary to change the organization of the components,

and the expertise of the implementation specialist is not enough, then you need to see the whole system, the formation of which the architect is engaged in. It is extremely important for him to understand the various programs, and how they can interact with each other, while deep expertise in all areas and the current situation in the teams is difficult to achieve, but there are leads for these teams. Once it was decided to make such large-scale changes and critical changes, the architect needs to work closely with business (stakeholders) so that the new system meets the expectations, and the transition is not so painful. Beginning in 1962, the Master Plan for Information System proposed defining the necessary infrastructure, analyzing the current one, identifying differences, drawing up a transition plan, and implementing the transition. Following the plan and its detailed description is a key requirement for a successful transition and achieving the final state of the system. Due to the heterogeneous nature of the implementation, in 1988 Stategic Information Planing divides the architecture into technological, system, functional, and data architecture. In 1992, layers appeared in EAP detailing: planning, a business layer, information (data, applications and technologies) and a layer of implementation and migration, which made it possible to speak their language with business and IT. Further, in 1995, to reduce the risks of changes and obsolescence of the requirements for the final system, TOGAF proposes to divide the whole process into parts ("cycles"), each of which will be presented by a separate

implementation project for a separate manager and combined into a "project program" for which the authorized manager. The interactivity of the architectural cycle itself is close to the spiral approach in development and also requires the invariability of the result in the project. At the current iterative cycle, you cannot accept changes from interested parties, but you can take them into account at the next iteration of this cycle. An important point is that TOGAF is presented as an architectural framework that requires adaptation and selection of the necessary elements, and not a ready-made process and set of rules for describing and following a plan. Which, on the one hand, discredits the idea of him as a silver bullet in the necessary set of hundreds of documents (artifacts) connected by a general application rule (architectural cycle), and on the other hand, shows his flexibility as akin to a set of programs in Linux. With the transition to cloud infrastructure, even if the company does not plan to use public clouds, it often seeks to keep private for standardization of management and application. Thus, from the development of solutions based on the cloud, the physical (network, server) and technological (virtual batch, container) are transferred from the application to the cloud ecosystem, leaving the business level, IT level and level for migration. An architect defends his project in front of a business to receive budgeting for its implementation.

DoDAF takes the concept of different points of view on the product, which can be especially when accepting a product made by customers, and the rest is aimed specifically at the

executive order, presenting a multi-level detail from the idea to the implementation of changes, which details the creation towards the transition to it. TOGAF is the successor to Zachman, and ARIS is very similar conceptually and schematically, so we will look at TOGAF and ARIS in general and focus on the differences.

For TOGAF, I recommend using the Archimade1,2 flowchart program, which can be downloaded for free on the official website. TOGAF assumes the stages in the development cycle (separate project): architectural vision (strategy), business architecture (business processes), system architecture (in the Archimate layer of elements – applications), technological architecture (technologies), applications (physical) and a group of stages migration (migration). TOGAF itself does not contain any software in itself, but carries only a description (framework), different programs can be used for visualization, according to this TOGAF carries only advisory character, and the program itself is single-user. ARIS is already a group of commercial products, the framework of which comes with a free version of the ARIS Express program (ariscommunity.com), and since 2010 with a cloud solution. ARIS contains layers: goals, concepts (regulatory documents), base processes (148 types of blocks), objects of business environment. ARIS is more focused on big business, as it contains many unified approaches, they will provide a joint development model (client-server application), an automation language, and various paid services.

Project success for the project manager and architect are different. So, a project manager, as a “foreman”, must form the processes in a team and interact with interested parties in such a way as to meet the border requirements, which are fixed in the charter of the project, and submit the project. By the boundary conditions, the overall functionality, development time with its cost, which is important for the customer (project sponsor), otherwise the project will have to close. Buffers are laid during time and cost when planning risks, and it is negotiated with users about functionality, which allows balancing within the boundaries of these indicators within the uncertainty (laboriousness can be more than planned, sick employees increase the duration of the project, the necessary functionality changes when user preferences change) and hand over the project to users. The project manager does not conduct business analysis to formulate requirements, but, like analysts, interacts with interested parties to adjust their implementation in order to facilitate the delivery of the project. Also, the project manager is not interested in the fate of the project after delivery.

The architect also works with interested parties, but he forms a vision of the system, which is formed in parallel with the requirements for implementation. An architect connects at the earliest stages in the formation of requirements that he is able to influence. Also, the architect does not leave the project after delivery, since the developed system, as a rule, continues to develop, and the project is just a stage of its development, which

promotes TOGAF in its architectural cycle.

Конец ознакомительного фрагмента.