TranzAxis and the payments industry

COMPASS PLUS TECHNOLOGIES

Payments industry evolution

The evolution of the payments industry to date can be loosely placed into three categories. For the purpose of this ebook we will call them eras. Each era has different technology and different approaches to technology itself. Eras do not have set boundaries; different countries across the world mature at different rates, as do market segments and market players, making it entirely possible for different eras to co-exist in the same timeframe.

To simplify matters, let's call the three main eras, pre-modern, modern and post-modern.

The pre-modern era

The era when electronic payments made their first appearance and the payments industry was born. At first, payments services were only available in financially developed markets and to a very small number of customers. Payment applications functioned on unique mainframes, software was enormously expensive and they required very highly skilled staff to run and maintain them.

The demand for payments services grew rapidly, but the technology of this pre-modern era prevented the industry from fulfilling these demands. This led to the first technological revolution and the transition into the next era.

The modern era

This era can be distinguished by the fast spread of electronic payments globally, with the appearance of private, local and national networks, and the rise of global networks as these smaller networks united. This growth can be attributed to access: the widespread reach of card acceptance and quicker card issuance turnaround. The industry grew quantitatively rather than qualitatively: the same services were available, but for a larger number of people.

To support this growth, a need for new technology arose. Mainframes were replaced by Unix and Windows systems; Cobol, Fortran, TAL, replaced by C and C++, whilst X.25 networks were replaced by TCP/IP. The modern era is one that embraced the mass distribution of standard services, and therefore the main requirements for these systems was accessibility: standard, not too expensive hardware, reasonably priced plug and play software and staff requirements sat at mid-level qualifications.

A typical payment application from this era can be described as a standard box you can move to another location, unpack and start issuing cards (a very simplified example, but prudent never-the-less). The potential for extensive development had been exhausted. Everyone had cards in their wallets, every shop had a POS terminal and every street had an ATM. There was a need for a paradigm shift, driven by the question: what's next?

The post-modern era

The transition to this era began in developed markets several years ago, and it will continue to spread across the world as time moves on. The main driver of development is the penetration of financial services into every aspect of people's lives; creating a digital economy. This era is defined by the hundreds of specialised products tailored to specialised segments of the market. Around more traditional institutions, FinTech start-ups appear, quickly developing their own niche and solutions for it in order to gain competitive advantage over their slower moving competitors. This era is defined by the personalisation of payments: no longer are you considered competitive if you are exactly like everyone else.

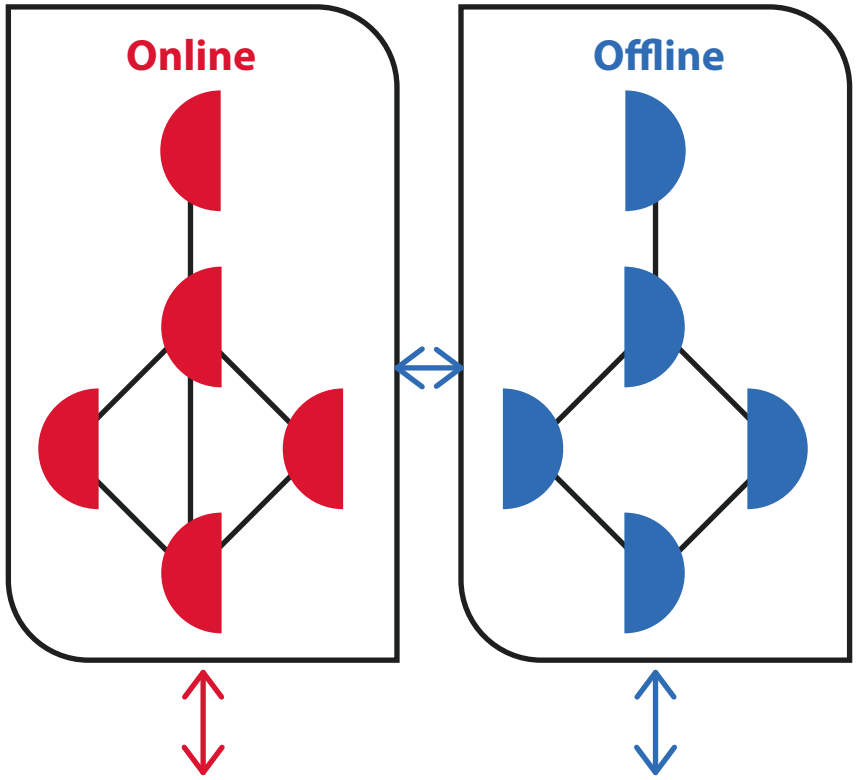
This era is led by the creation of unique financial services and, therefore, the main requirement for payment systems is flexibility. The time for "boxed" solutions has passed, and the time for open platforms has arisen.

The evolution of architecture

As we transition between the eras of evolution in the payments industry, we can clearly see the fundamental changes to the approaches for building payment services. The driver of these changes has always been to meet the new requirements that each era has presented. It is the development of the technology that has made these changes possible.

In terms of architecture, the pre-modern era has been analysed and dissected enough. We all know the advantages and disadvantages that mainframe systems afford, so let's move on to take a closer look at the modern era.

The most obvious characteristic of payment services architecture for this era is the very strict separation of offline and online systems. This division is largely due to the fact that, during this time, the architects who built the foundation of the software were limited by the development of information technologies and what general computing allowed. It did not allow for the effective combination of real-time and batch load in one system. An example of the typical architecture for this era can be seen on the right.

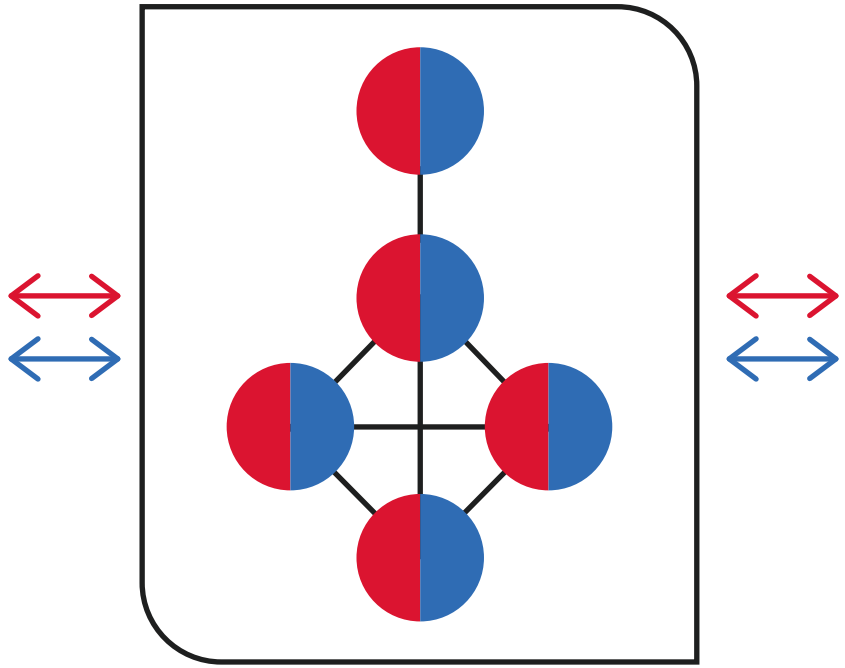


It's important to note that the line of separation here is not based on the functional components, but is specifically dictated by the technical limitations. As a result of this division, the business objects (cards, accounts, customers, contracts, etc.) are split based on technology - not business logic. Each object lives concurrently in two or more systems, whereby no single system knows the actual up-to-date information about the current state of the object: no one system has full visibility of the object.

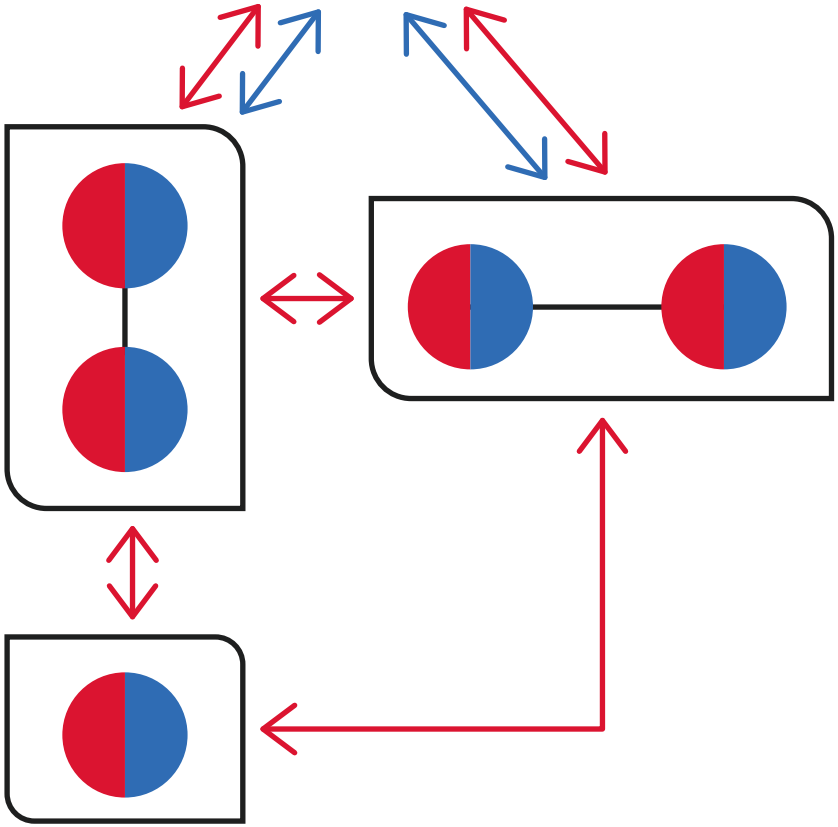
The division into online and offline systems was justified and tolerated at the beginning of this era, but gradually became a hindrance over time. Today, customers want everything right now, and expect all relevant information to be available 24/7/365, which is completely impossible if the information is spread across several systems that only talk to each other via file exchange.

Another distinctive characteristic of payment system architecture in this era, is poor connectivity: separate components interact with each other via narrow interfaces with limited message types using proprietary non-standard protocols, and the prevailing method of information exchange is files. As time progressed, with the development of a larger variety of payment services and the need for their integration into the wider context of the digital economy, these integration flows have become increasingly more complicated and mainly online.

As we move into the post-modern era, the development of information technology has allowed the move away from the separation of systems (into online/offline) and has enabled the architecting of systems based on a single responsibility: whereby each business object lives in one system which has a full view of all object-related information and can perform all operations specific to it (of course this is within the remit of one business domain).



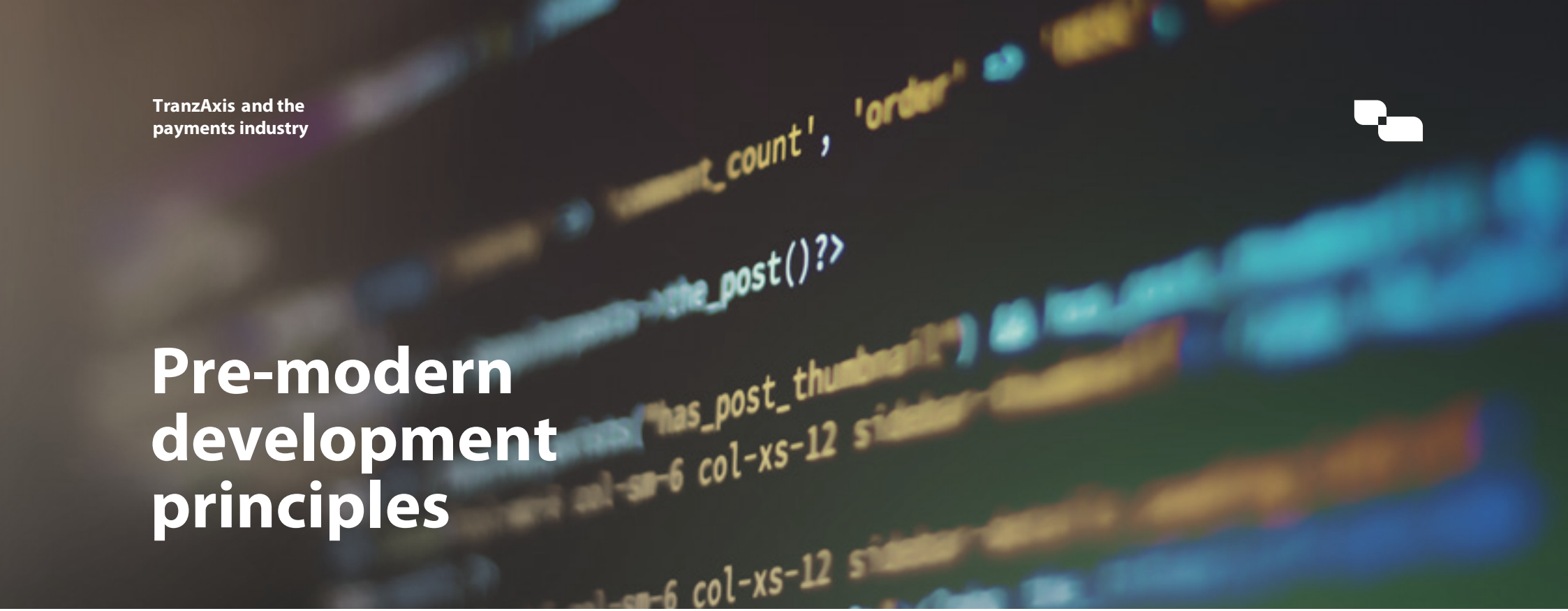
This doesn't mean that the modern architecture implies the use of a single, enormous monolith. In large FIs it makes sense to divide the monolith into several services (micro services), but this separation is not determined by technical limitations, but by functional charateristics.



In order to function effectively in such a SOA architecture, components should be designed in a specific way. It becomes critical to have a fully-functional, reliable, well-structured and documented API that supports modern standards and technologies for integration. In modern systems, APIs are not something you bolt on whilst the system core is developed, they are one of the most important architectural components embedded into the DNA of the system.

The evolution of development methods for payments services

The payment industry has been developing rapidly throughout its existence. However, the methods for developing and implementing new innovations in each era are fundamentally different.



Pre-modern development principles

At the conception of the payments industry, each new payment service was unique and each payment system was bespoke and created from the ground up. It very quickly became clear that building each system from scratch was both very expensive and inefficient. It was then when the first, replicable product was born - ACI BASE24. This was a massive leap forward for the foundation of the whole industry.

BASE24 came with the source code and many users of this product programmed new modules on top of it or even modified the core. At first glance, this allowed FIs to develop their services independently without relying on the vendor. However, this development method had and still has a fundamental problem: the complexity of product upgrade. When a new version of the product is supplied, the users have to rebuild and check every one of their developments and customisations. Each upgrade suddenly becomes a long, complicated, expensive and risky project, and the further you develop and customise the product, the more it costs, until eventually it no longer makes financial or business sense.

Modern development principles

The initial versions of software products of this era were rather limited in flexibility. All the user could really do with them was change the pre-defined parameters. As services developed the number of parameters increased exponentially and it became apparent that further development along this route was impossible. Some vendors decided to give source code to their customers and let them make changes to it themselves. It's clear that this cannot be considered a great idea. After several sales like this, the vendor ends up with several versions of the same product, that each need to be supported separately. No vendor can actually afford to do this. The customer has a clone of an old version of the product that they have to support and develop by themselves. This approach is only viable for the largest and richest FIs, but even then, it becomes increasingly expensive year-on-year.

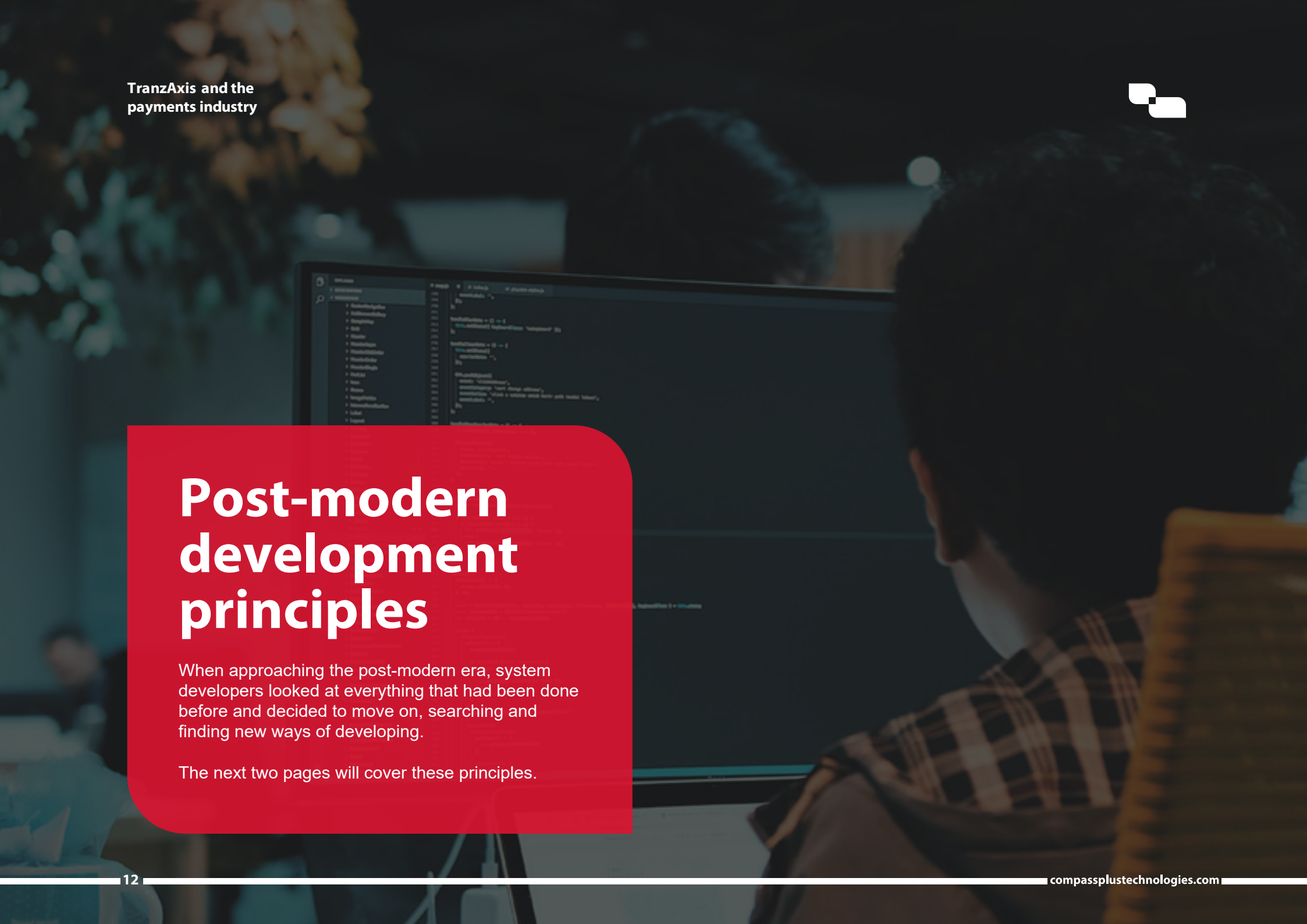
Some vendors found a better way of developing their systems. Their customers gained the ability to extend and modify the product using various user scripts, user functions and user exits (we call this algorithmic customisation). At the same time, the core of the product remains unchanged and common for all customers: the vendor can effectively develop it, and customers deploy new versions relatively easily.

The idea of algorithmic customisation has been quite successful. It has enabled the industry to progress for many years. However, it's potential is not infinite and has fundamental limitations:

- The user can not change the core of the system and therefore is forced to cram innovation into the object model and the transactional engine of a standard product. Almost every product of this era is built around one main object: the card. For these products, the card is the only payment method; the only means to identify and authenticate the customer and the only way to access the card account. The execution of card transactions is the only way to make a payment. With time, these limitations have lessened, but it's impossible to completely get rid of the inherent characteristics that are built into the systems DNA, you can only lessen it

- As the industry developed, non-card payment methods and transactional scenarios arrived. However, the developers of algorithmic customisations have to emulate all of this in the old card language. Implementation of new functionality in these conditions isn't simple. The system turns into an ugly tangle of low-grade substitutions that grow more expensive to maintain each year

- From the point of view of today's developer, algorithmic customisation technologies are last century. IDEs, tools for analysis and code review, version control, automatic testing delivery and implementation, etc. are all either missing or not fit for purpose



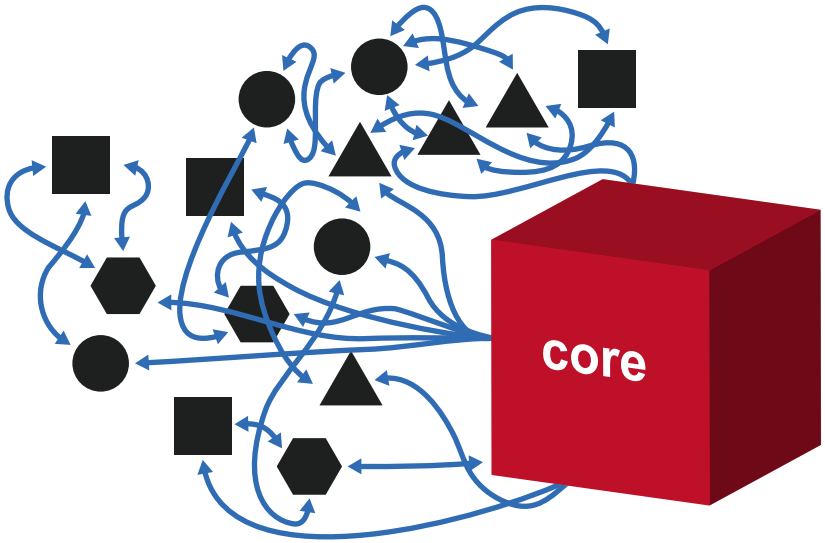
Post-modern development principles

When approaching the post-modern era, system developers looked at everything that had been done before and decided to move on, searching and finding new ways of developing.

The next two pages will cover these principles.

Old core, new periphery

Many traditional FIs, well established in the market are forced to follow this path. In the centre of their payment infrastructure, playing the role of transactional core, are the systems that were built in the early modern/pre-modern eras. These systems are exhausted in terms of development potential. Therefore, the majority of innovations are developed outside the code in the peripheral services.



This approach has clear disadvantages and limitations. First of all, the core is old and it significantly hinders the implementation of new functionality. A legacy system that was most likely built during an era that was pre-internet and smartphone, where the only payment method option is a card, cannot adequately reflect the real objects and functions of the modern payment world. Developers therefore have to constantly replace the real payment operations with fake card transactions, and otherwise mimic the real world in order to trick the core, as it is unable to understand the payments world as it stands today.

The old core also prohibits the ability to solve technological tasks. Using modern integration technologies, monitoring, virtualisation, cloud deployment, automation, CI/CD, - all rests against the limitations of the old system and is impossible to use. The presence of the old core makes it impossible for FIs to fully take advantage of the technological advancements of the IT industry.

FinTech start-ups with an external processor

In this instance, the start-up independently creates their own customer facing services and uses an external processor to access the larger payments infrastructure to perform actual financial transactions. This allows FinTechs to concentrate on their areas of competency and to give the “boring” part to someone else. The problems to this approach are obvious: traditional processors cannot offer the advanced APIs they require, neither can they develop and maintain them as fast as required. Many processors are oriented to providing standard mass market services at competitive prices and either cannot, or will not, provide each of their customers unique capabilities and services.

FinTech start-ups with full development cycles

Here the FinTech independently develops and implements all of their main systems, starting with front end applications for customers and ending with a transactional engine. Potentially this approach can give FinTechs maximum freedom to innovate alongside vendor independence. Many have tried to take this route, but only single digits have been successful.

The development of payment applications is a very complex task that requires very unique experience and qualifications. The global market has a finite number of specialists that are capable of building payment systems from scratch. Development from the ground up takes years and has insurmountable risks. From the time it takes to build a full payment application from scratch, the potential is high that the initial idea will already obsolete and no longer make business sense. By and large, this route is only applicable to giants with near unlimited resources such as Apple, Google, Facebook, that finance the payment part from a completely different business domain.

Introducing TranzAxis

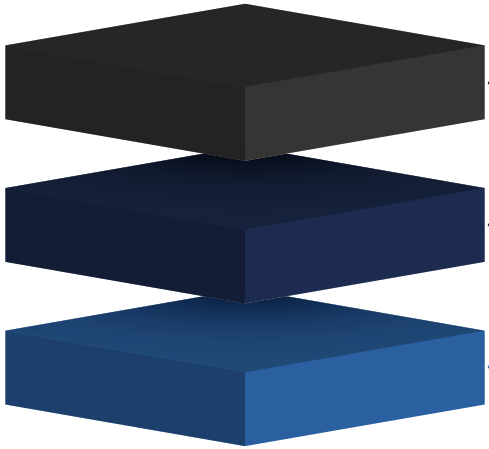
As demonstrated above, for any FI, both the use of the old core and the independent development of their own platform can be unsuccessful strategies.

You cannot use a system whose skeleton was created at the dawn of the payment industry as a payments platform today. Such a “box” is not able to adequately represent the objects and functions of a modern payment environment, is not able to effectively interact with other systems in a service-oriented architecture, and does not do well in a dynamic cloud environment. And most importantly, does not enable rapid development.

It is not practical to create a payment platform for an individual customer. In addition to the associated problems of cost, duration and risk described above, this will inevitably limit the flexibility and universality of the system. The platform must be initially designed and developed as a replicable product suitable for solving a wide class of tasks for a wide variety of clients. Only then, will it have the sufficient margin of flexibility and strength to guarantee customers’ unhindered development for years to come.

At Compass Plus Technologies, we decided to give financial institutions another option - to use our TranzAxis product as a platform for innovation. The TranzAxis platform (in addition to algorithmic customisation) offers a new way of development: developing a custom layer.

The system, built on the TranzAxis platform, is a stack of several layers:



The TranzAxis layer implements all the objects and functions necessary for building payment applications: a transaction engine, a financial accounting mechanism, cards and other means of payment, customers, accounts, interfaces, cryptography and much more. TranzAxis is supplied and maintained as a replicable software product.

On top of the TranzAxis platform, our customers themselves, or us, based on their requirements, can create and develop their own layer, where they can implement any unique features they want in order to offer competitive and differentiating payment services.

This development method removes the problems described above and has distinct advantages:

- There are no restrictions on what can be done in the custom layer. Here FIs can create their own objects, transactions, interfaces, batch procedures, workplaces, reports, etc. Actually, here FIs can do everything that we can do in TranzAxis, without exception or limitation. They have access to all the development tools we have and use
- Any customer who does their own development has direct access to the richest API a platform has to offer. The customer does not have to "reinvent the wheel", they can concentrate on solving their business problems and not waste time re-implementing standard functions
- Development tools are supplied as part of the platform. These are present-day industry-standard tools, exactly those we use when developing the platform: modern IDE, testing, integration, delivery, maintenance tools
- A customer doing their own development can be assured that after installing a new version of the platform they will not have to redevelop and test their code. The platform guarantees compatibility which is technologically enabled, and not prone to human error

TranzAxis and the payments industry

Designing a payment platform is a very significant task with the need to find and constantly maintain a delicate balance between conflicting requirements.

- The platform should enable the quick creation of new services, but at the same time strictly meet the requirements associated with security, performance, fault tolerance, scalability, compatibility with industry standards
- The platform must be extremely flexible, but it must not be a do-it-yourself lego-style design. It should make it easy to implement standard functions using standard tools, and enable the FI to focus on the development of innovative functions
- The development tools provided by the platform should allow any idea to be implemented. However, the implementation of a new function should not start from scratch every time; all objects, functions and services in the platform should be available to the developer
- Customers need the opportunity to freely create their own differentiating services on our platform, whilst continuing to receive platform updates without compromising their own developments
- Development tools should, on one hand, be modern and recognisable to developers, and on the other, focused on being fit for purpose for the payment industry
- The developer must have the freedom of creativity, but the system being created is business critical and must be protected as much as possible from human error

Our award winning, open development payments platform, TranzAxis, achieves all this and more

compassplustechnologies.com