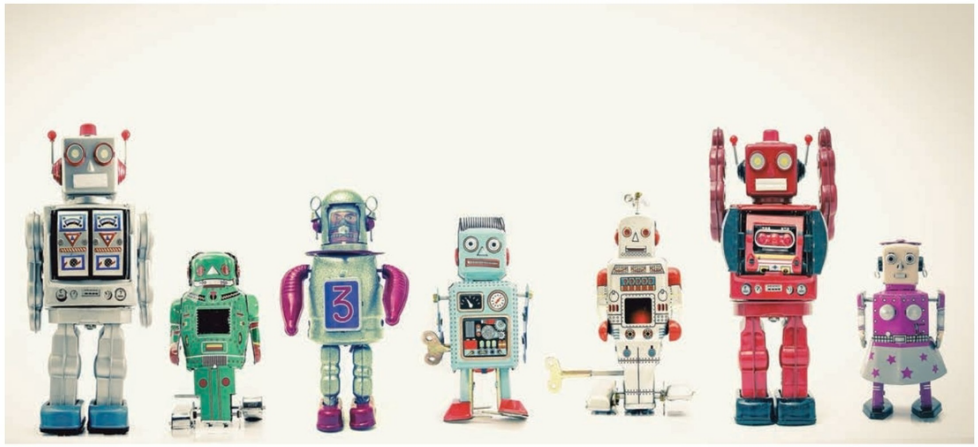

THREE LAWS of Softwaristics

Building on the fundamentals means easier evolution

In 1942, Isaac Asimov introduced the sci-fi literature world to the Three Laws of Robotics:

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law. movement of its electrons.

Pascal Polverini is a software architect at Fresche Legacy and co-authored the latest IBM i modernization Redbooks publication.

The Three Laws of Softwaristics

Thinking about software application development, debugging, maintenance, deployment and overall application evolution, I propose that the same law structure be applied to software:

1. Software may only use open standards or, if nonexistent, use interoperable formats.
2. Software must use a multitier architecture, except where such architecture would conflict with the First Law.
3. Software must centralize its logic, as long as such centralization does not conflict with the First or Second Law.

Law 1: Open Standards

This law applies to software languages as well as data formats. If the technical specification for the language or the format is public, then it's open.

Ruby and PHP are open-standard programming languages, XML and HTML are open-standard formats, and HTTP is an open-standard protocol. And while not truly open, Java* and RPG follow standards as well, rather than introducing cost and risk by reinventing the wheel. This also introduces risk for the sustainability of the software, as whoever uses it after you may not be able to maintain it. However, when standards are followed, you don't have to think about it anymore, as the standard secures the sustainability of the software as technology evolves. The technological evolution will either change the standard and provide a means of conversion from the old version to a new one or offer new products that will integrate the existing standard.

A good example of this would be Open Office. To make it widely used by the community, the developers created a converter from Microsoft* Word. The new RPG IV standard is similar, with the command CVTRPGSRC to convert RPG III to RPG IV.

In the absence of open standards, you may use interoperable formats—not so much for programing languages, but for data storage and exchange. Programming languages can be platform- or device-specific or cross-platform or -device, and that doesn't mean you have to choose between integration and deployment capacity. What is determinant is mainly the purpose of your app. If it's destined for the cloud, you may look for a multitenant back-end system and options to also enable offline syncing from your front-end device. In any case, even if you choose platform- and device-specific languages, you may use a suitable file format for data preservation and sharing. Therefore, deployment won't be an issue now or later.

Law 2: Multitier

Tiers could differ in naming, but the standard multitier model comprises three core tiers in three respective blocks: The data tier, which stores the data; the application tier, which processes the data; and the presentation tier, which presents the data and enables input.

For example, Excel spreadsheets include all three tiers in one single block, which is bad from an application perspective. This is why the .csv format was invented. It represents a data tier block separated from the presentation tier block. Text messaging uses two tiers in one block: presentation and data. More sophisticated software, like ERP or CRM, will have all three tiers in different blocks for databases, business processes and different presentation channels.

Having an app divided into multitier blocks allows each tier to be separately developed, tested, executed, reused or substituted.

A tier could be logical or physical. Physical tiers are entire blocks (e.g., a database file or a cascading style sheet), and it's easy to substitute a physical tier with another one.

Logical tiers can be considered as a layout within their environment. For example, DDS and RPG each comprise two tiers, but they have a layout in common: the buffer definition. Within the DDS, this layout is mixed with the UI presentation layout (another layout). While this can be seen as advantageous, as you get a centralized buffer description for both the RPG (application tier) and the file device (presentation tier), it would be better to have two distinct blocks—one for the buffer description and another for the presentation layout—even if they are within the same container. This way it becomes equivalent to a true tier block that can be easily substituted. (Do you wish DDS were XML? Read "Life After DDS" ibmsystemsmag.com/ibmi/developer/rpg/oamos-intro.)

A well-designed app will use different physical tiers, but also logical tiers.

For instance, in the application tier, a logical tier might have a distinct function for a validation process and, furthermore, this distinct function could also be placed in a separated physical module. The same principle can be used for data in the database—a distinct file or aggregate for NoSQL could be created.

Finally, a multitier architecture will increase the agility and sustainability of changing your app with market technology or device evolutions.

Law 3: Central Logic

This law concerns logical rules, (aka business rules). You can have absolute or relative rules.

An absolute rule can contemplate the type of a data field, its validation or its correlation to another field. For instance, a field can be set to always display a date or a number, or to always be validated in a unique way. It can also be set to not exist if it's not defined in a header file.

A relative rule can depend on the user login, contextual data or the environment. For example, if I must show a grid made of five columns on a GUI, I may show the entire grid on a browser desktop but only one column on a smartphone.

What matters in our open-standard and multitier premises is that absolute rules are defined at the data tier or, if not possible, at the application tier. Relative rules are defined at the application tier and, if not applicable, at the presentation tier.

The goal is to write any business rule once and on a single place. But if, for application reasons like speed transaction, you need to replicate the business rule in different places, you can still integrate a procedure that inherits (or references) the rule from a central place.

Finding Meaning in the Fundamentals

When you use fundamentals, you gain coherent results—but more than that, you gain durability. Because your model won't be the consequence of a casual "context," it's in harmony with the overall ecosystem; therefore it persists through evolutions. Actually, we can say that the fundamentals are the ecosystem. IT is not a natural science; therefore its ecosystem can change, but within it we can still look for fundamentals. P