The Macrotheme Review
A multidisciplinary journal of global macro trends

Cost equivalent of complexity measure - Measure complexity and utilize the result in financial planning

Tamás Nacsák
Enterprise Architect at Telcotrend, MSc student Óbuda University – Keleti Károly Faculty of Economics

Abstract

In the course of my career at large enterprises a frequent issue mentioned was the operation, processes, systems are too complex. The topic was only based on intuitions, mentioning the complexity as the root cause of problems like slow-running processes, large number of errors, slow and expensive developments, or high operating expenditures. The most interesting part of the situation is that lots of rules and directives, complicated processes at first sight or large number IT applications enwreathed to endless spaghetti are all can be found at these companies. The attempts I saw in the course of the simplification ambitions all tried to lower the numbers of entities (either rules, processes or applications. The most impressive – but unfortunately still unsuccessful! – activities tried to replace many small IT applications with a new massive integrated solution. Usually, these projects consumed huge amount of money and they did not live up to expectations either if they reached their target at all. The experiences suggest that complexity is not simply described by the above-mentioned components (e.g. number of IT applications), the solution could be somewhere else. Having no proper measure for the complexity it is not possible to go for optimisation! The article shows a great method to measure the complexity, what is simple to use. On the top of the complexity measure it is necessary to confirm the correlation between the complexity measure (called Standard Complexity Unit – SCU), which would be then solve in a single step the investment calculations for simplification targeting projects. Readers can learn the details of the proven SCU calculation method I used at many companies, what is the basic of the research target. On the top of SCU we will then look for the possible ways to find the cost equivalent of SCU. Based on SCU we make a step forward to find the Complexity Cost Equivalent (CCU), which hopefully brings us closer to understand the case of immortal temporary solutions or simplify the simplification endeavours.

Keywords: complexity, simplification, financial planning, cost of failure, modern project management

1. Literature review

Enterprise architects are faced with many architectural options. Without scientific measures, these decisions are necessarily subjective. This results in lots of arguments, but few definite answers. Until now. The article describes the use of a simple, clear, and objective approach to calculating the complexity of an architectural proposal. These calculations have the following important characteristics:

- They are precise and independent of the person making the measurements.
- They are definitive and easily validated.
- They are easy to determine and require no tools.
- They can be calculated very early in the project life cycle.

The referred materials show a number of architectural projects that have had their complexity calculated (described by Standard Complexity Unit – SCU). The results suggest that these complexity measurements are highly predictive both of project development costs and project maintenance costs (Csiszárik-Kocsir – Varga, 2016a). This further suggests the exciting possibility that complexity measurements can be a useful approach for deciding between alternative proposals. These results provide validation for many of the ideas proposed by Roger Sessions in his paper, The Mathematics of IT Simplification. The repeatability of the calculation requires an information store, an architecture repository, which is based-on a data model, which is capable to map the 360° view of the enterprise. Finally, the article describes some potential approaches to find and confirm the cost equivalent for the SCU. Three great things: information repository, complexity measurement and cost equivalent. The first two are already proven in practice, the third is still hypothetic but is so promising.

2. Methodological background

Measuring complexity

SIP and the complexity measurement are a simple, clear, and objective approach to calculating the complexity of architectural options (*Sessions, 2008*). These calculations have the following important characteristics:

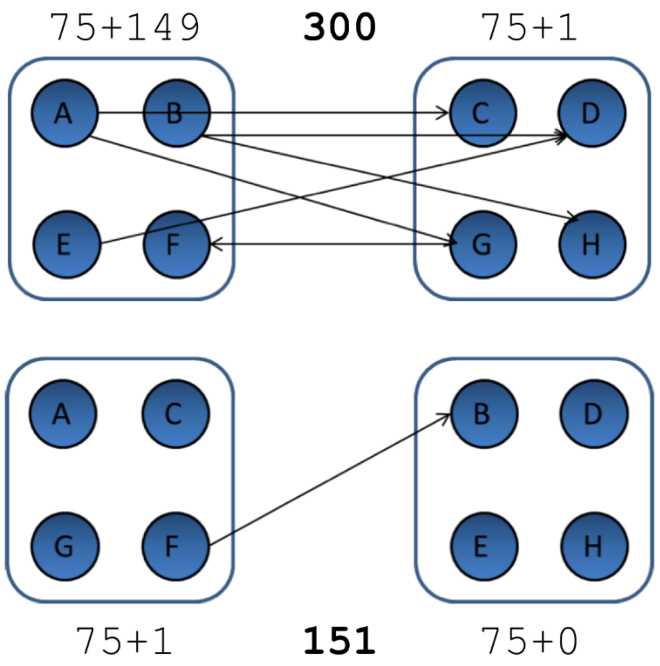
- Precise and independent of the person making the measurements.
- Definitive and easily validated.
- Easy to determine and require no tools.
- Can be calculated early in the project lifecycle.

I hope you are hardly waiting the details after this long introduction. In simple word (how else!), collecting the number of functionalities in "boxes" and the dependencies/interface between them would be substituted to the following equity:

$$\boldsymbol{C(N,M) = M^{3,11} + N^{3,11}}$$

The argument calculated from the number of functionalities (M) called Functional complexity and the name of the other is Coordination complexity calculated from the number of dependencies (N). Finally, the sum of these two arguments gives the complexity count (or as Roger calls SCU - Standard Complexity Unit) of the "box". The box is usually an application or a functional module of that, depending on the level you would see the results. In my experience I am using the functional module level. The following figure shows a basic example, where the boxes represent functional modules, the blue discs are functions, and the arrows are the dependencies.

Figure 1.: Alternative partitions



Source: The Mathematics of IT simplification

In this example, all modules have the same functional complexity level, the difference is the coordination complexity. Jump into the details, describing the assumptions and inventions behind! All we agree that dependency between the number of capabilities or dependencies and the complexity is not linear. One hand this is a feeling by heart but also confirmed by the real-life experiences. The question then is the exponent. Roger suggest using Robert Glass's number what is calculated from the assumption, that 1.25x more functions doubles the complexity. Getting more details, please read the book Facts and Fallacies of Software Engineering by Robert L. Glass (*Glass, 2003*).

Now we arrived the first real challenge: how to collect the functionalities and the dependencies. To feel the size of the challenge, we have to define the real meaning of functionality and dependency.

- **Functionality:** when we say functionality, it means atomic function of a given application to ensure that valid numbers are used in the equity.
- **Dependency:** it covers the connections between applications by a common word the interfaces.

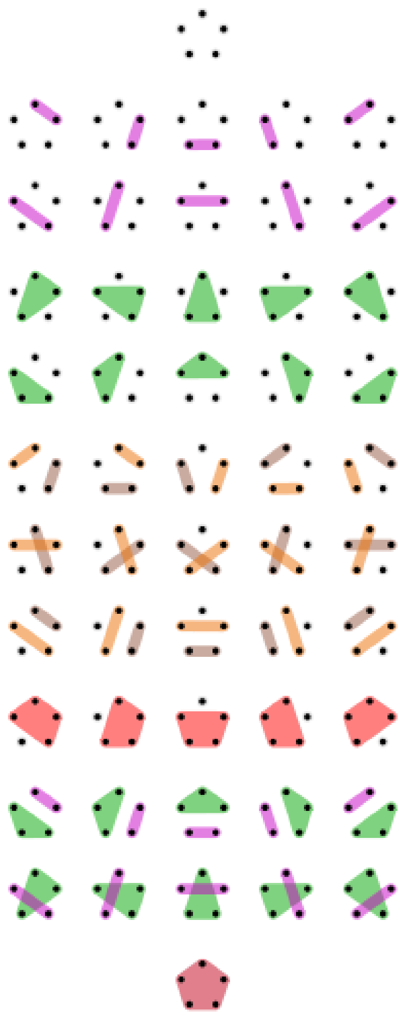
The dependency is the simpler topic, since more or less everyone knows the interface between the applications, while the linkage of the interfaces to the functional modules can be a bit more challenging. Turning to the functionality we will face the real issues, since you have to ensure that all the functionalities are atomic what you or your team collect about the applications. Let's see an example: assuming you have a webshop which has checkout function at the end of an order, you may record checkout as an atomic function. It is fine, if the actions running behind (user identification, invoicing details, delivery setup, payment) are not used separately - if they are, you have to use the actions behind instead and then do not record checkout additionally! The other challenge is, staying at the example of the webshop if in case of web orders checkout is atomic, but in shop order handling application it is not. You as architect has to aware to split webshop checkout to the same atomic parts as in the case of shop order application! It is crucial to get comparable SCUs for all applications! Having the same granularity of each function around the scope of the complexity calculation requires strict documentary disciplines, right principles to identify atomic functions and last but not least quality assurance from EA team.

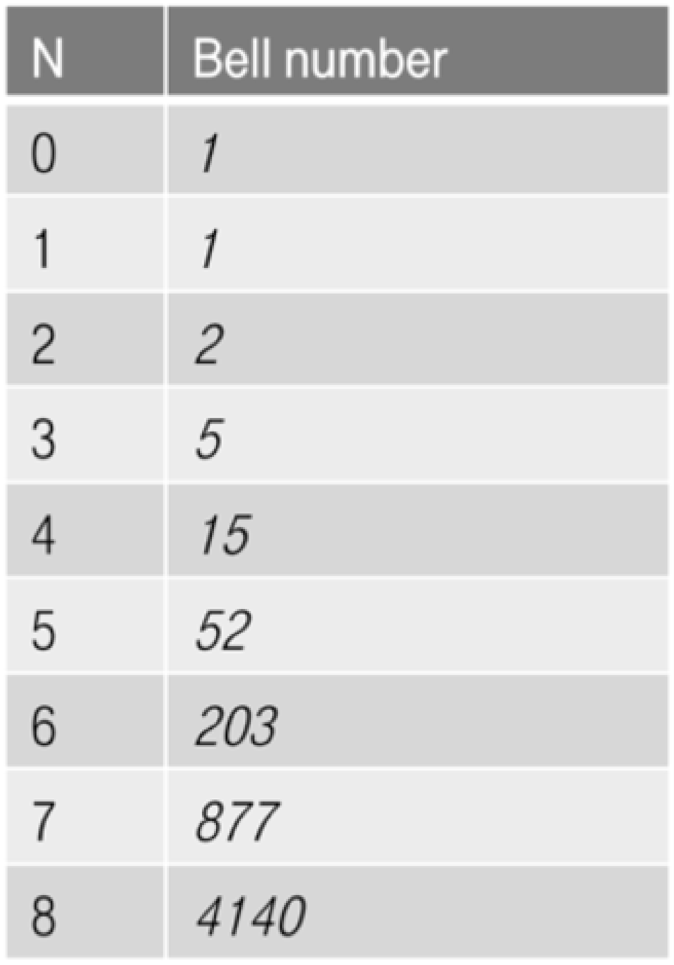
The simplest possible architecture - introducing SIP

Having no exact method to reach simple solution all of us who is responsible for designing solutions using the strong faith, believing that our result is the best. Sometimes it is true but sometimes not - and no one knows what the current case is. The complexity measurement referred above helps to evaluate a given situation and its alternative options, but the next question is if we really evaluated all the potential options? This generates the next problem. Ensuring that we evaluated each option we have to know how many exists. Luckily it has the mathematics, which helps us.

Let's run through a theoretic example! The exercise is to implement a set of functions on the way that is the least complex, while some of them should be included into existing applications. Using other words, we should find the best way of boxing them. How many options we have to do it? The answer is the Bell number which counts the possible partitions (referred as boxing before) of a set.

Figure 2.: The shape of Bell number



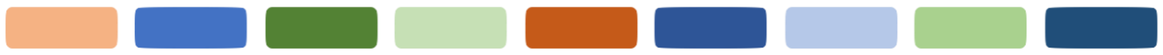


N	Bell number
0	*1*
1	*1*
2	*2*
3	*5*
4	*15*
5	*52*
6	*203*
7	*877*
8	*4140*

(Source: The Mathematics of IT Simplification)

The picture above shows that 5 elements have 52 different partitions, while 8 elements have 4,140! Our example contains 12 functionalities to implement, which means 4,213,597 alternative options. It is impossible to calculate all of their complexity. We need a guideline, which drive us to the simplest solution! This is SIP - Simple Iterative Partitions invented by Roger Sessions (Sessions, 2011). What should we do with this large number of options? Practically nothing but we have to recognise the hunger for a method which does not cause exponential explosion of resource needs as the set of functionality increases. The algorithm is nothing else but the automation of the *similis simile gaudet* by other words: like attracts like. In the course of designing architecture, we tend to collect functions that are similar (equivalent) to each other - the challenge is the meaning of similar, we are going to come back to it. Let's see an example, where the task is to group the similar colours together. The initial set of colours, representing business functions to implement are the following. Pick a random element from the list of functions.

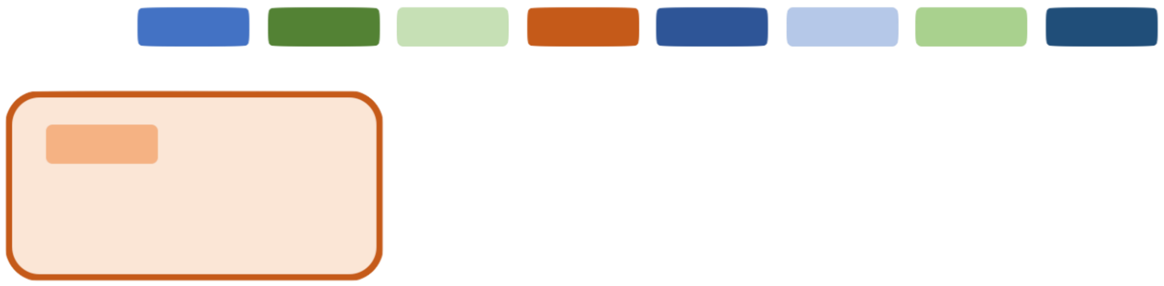
Figure 3.: The list of functions to be partitions



(Source: Author's own)

We did not collect colours in advance, therefore we do not have "containers", applications, functional modules, etc. The first step is to select on coloured box from the set above. What we will do, to look around if we have any existing container and if there is no one (since we just start) define a new one and drop the box into it.

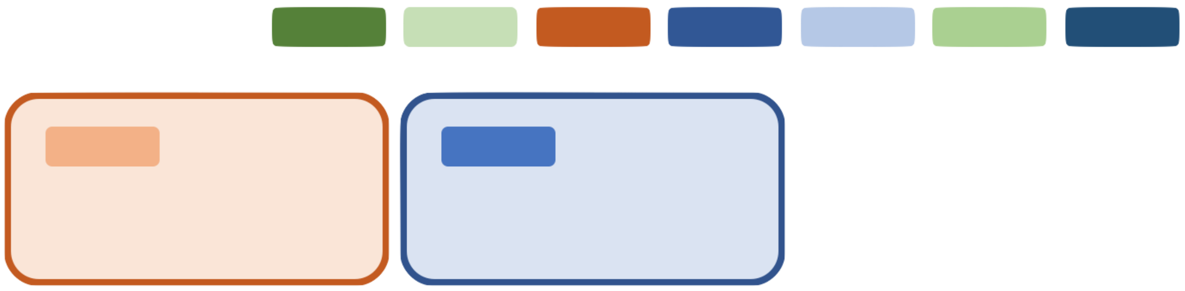
Figure 4:. Defining the first partition



(Source: Author's own)

The next step is to get the second box and check if there is any existing container. We will find the Orange one, but as it is not blue, we cannot use it (not similar!) therefore, a new one is needed.

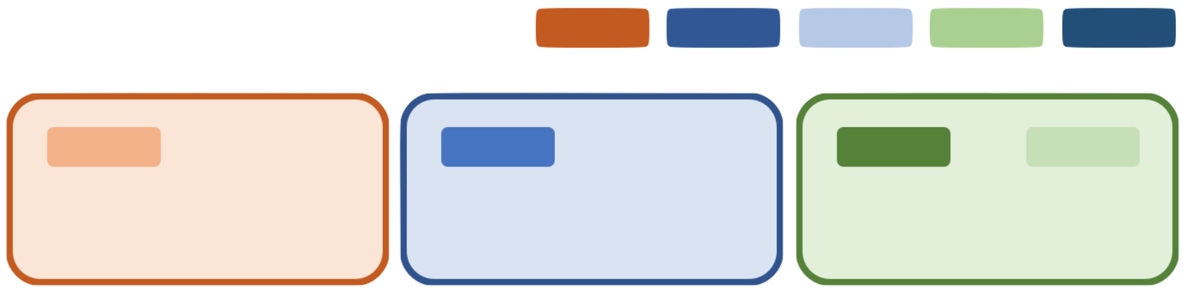
Figure 5.: The next partition



(Source: Author's own)

Continue the process on the same way, we will define a green container at the third step. Reaching the fourth box and checking the existing containers we will recognise that we already have one which contains similar element(s) then that box should be included also.

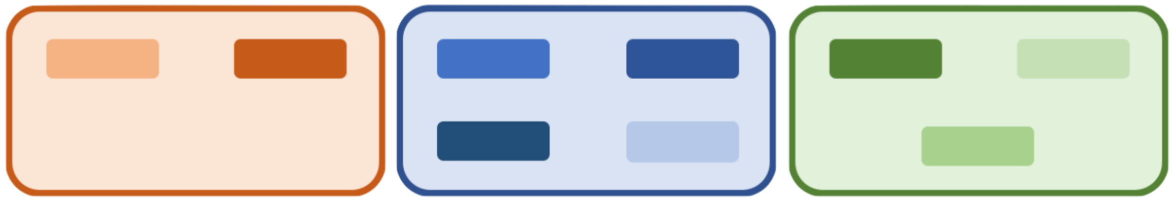
Figure 6.: The result of the next step



(Source: Author's own)

We have no other task no to continue the pick, search compare, include process till the end of the initial set of boxes, which will result the following architecture.

Figure 7.: The result showing all partitions



(Source: Author's own)

Recognising that one colour is similar to each other seems to be simple in the example, but if you try to imagine the full colour-spectrum we will agree that it can that hard as sometimes defines the best place of a new functionality in the architecture.

This approach works fine either having small or large amount of business functions to place. The difference on workload is linear, means the double of requirements is the double of resource to allocate. It is fine. The problem will be in large programs on the other hand that the collection and finalisation is a never-ending story. Assuming that everything will be collected in a waterfall approach is the sure-to-fail method. Luckily having enough requirements collected covering the whole spectrum of the project/program, the process guarantees that you will start implementing the right architecture! What can happen if you recognise a new requirement? There are only two cases:

- it will fit into an existing "container"
- a new container will be defined to solve.

You never have to remove or split partitions, either to move one function to another place! Never ever! Anyone who face restructuring of an inappropriate architecture any time rather in the middle of a running project will appreciate it!

Architecture knowledge base

Each knowledge base about architecture faces the same challenge: find the balance between details and accuracy. The less details would mean simpler maintenance of data while it certainly less usable. If you have limited resources for future maintenance, you have to make compromises. This article shows the way you can built up-to-date usable knowledge base with no bad compromises about scope and usability. Correct scoping, proper data model and architecture management tools are vital part of successful implementation. These will ensure the usability and finally the use of the knowledge base. Building continuous co-operation with your enterprise your knowledge base and consequently the architect team will be a trusted source and a trusted partner for everyone! The trust you earned enables one of the critical preconditions of the architect's main target: you will maximise the ability to change! (AEA US, 2017)

3. Results

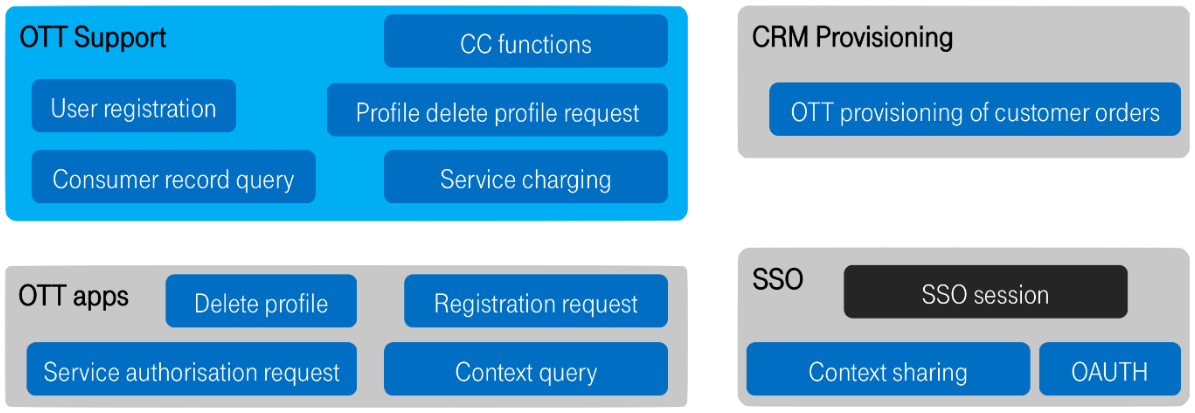
Solution planning with SIP

A practical example was to design a solution for OTT supporting environment (OTT stands for over-the-top services. It is the definition by telco companies for the ones like LinkedIn, Apple Music or the well-known Spotify) integrated into the life of a telecommunication company. The OTT solution has to have the following capabilities:

- User registration
- Registration request arriving from OTT providers
- OAUTH based authentication
- Delete profile
- Profile delete request arriving from OTT providers
- OTT provisioning of customer orders
- Customer care functions
- Context sharing between online pages or apps
- Context query of the used services
- Consumer/customer record query
- Service authentication request
- Service charging

Turning back to the business functions listed above the solution using SIP is the detailed in the following drawing. The light blue container is the new application, while the light-grey ones are the already existing applications. There is one dark-grey box in the SSO application, which represent an existing function, the SSO session handling which leads OAUTH and context sharing to be taken there. OAUTH is a "sign-on" function and context sharing between apps require similar kind of session to handle as SSO.

Figure 8.: OTT supporting solution



(Source: Author's own)

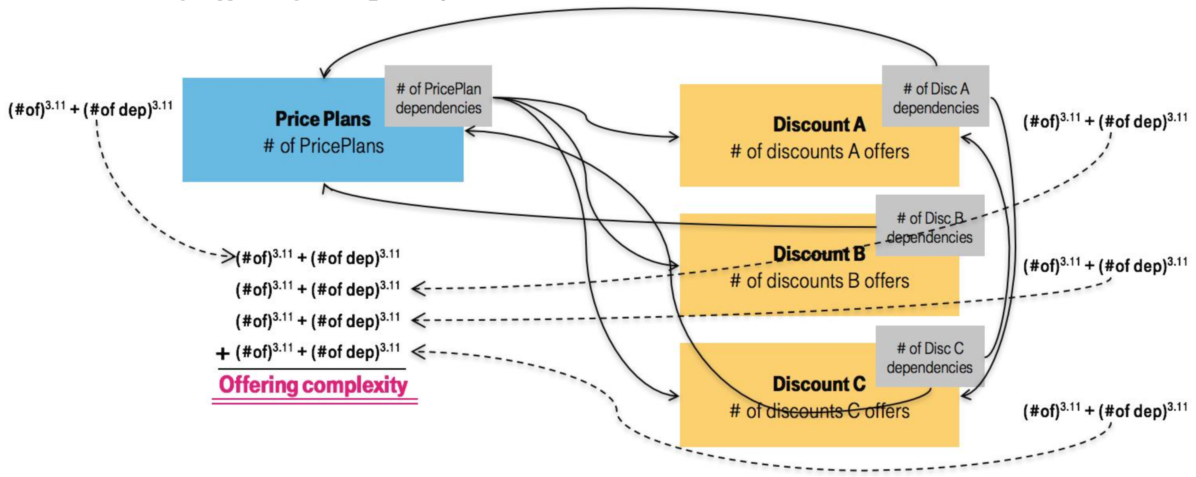
CRM provisioning application handles each kind of service provisioning towards network elements, OTT services are just the same as anything else from this prospective: best place found. OTT apps have to handle profile registration and delete and the authorisation to access the given service there. Context query is about the actual service use that means it is "equivalent" having the tight relation to other functions here.

The application called "OTT support" above, is still in operation. Following the original design patterns, the application first got a new function set called rule engine, which is a cart-based ordering capability, handling business rules. At this moment the application handled some thousands of customers and some dozens of business rules. Later it was extended to handle the entire product portfolio of the company, now it handles 5 million customers and beyond 1 million business rules! The lowest complexity makes the solutions antifragile (Taleb, 2013).

Product offering and product portfolio complexity

The most frequent criticism about product portfolios is: too complex! Based on the articles, which describes the problems of measuring complexity in general take a look to the product portfolios. A marketing manager defined me in the past: "Making a simple portfolio is an easy task; define a profitable one is the same. Creating one, which is simple and profitable is almost impossible." I do not know the medicine of the problem, but the method described here will help you to simplify using an exact measure same as financial calculations. Referring back to the complexity calculation above, here M means the number of substitute products in the offering, while N means the number of dependencies between the products. The following figure shows an example about a telecommunication offering, which contains price plans, aka tariffs and discounts. Calling back Roger's approach for the calculation, we had to identify "boxes" and "lines". The boxes contain the given products, aka offers which can be sold separately to the customers, but only one product can be sold from one box. As usual using this calculation the number itself will not tell us a clear message. The result can be some hundred or some million, depending on the number of M and N. The reason to calculate has two indirect results. First, two offerings will be comparable, which is good. The more important other end is that the calculation count will help to start to simplify the offering! The effect of changing dependencies, decreasing the number of offers inside a group (box) will be present there.

Figure 9.: Calculating offering complexity


$(\#of)^{3.11} + (\#of\ dep)^{3.11}$
Price Plans
# of PricePlans
# of PricePlan dependencies
Discount A
# of discounts A offers
# of Disc A dependencies
Discount B
# of discounts B offers
# of Disc B dependencies
Discount C
# of discounts C offers
# of Disc C dependencies
$(\#of)^{3.11} + (\#of\ dep)^{3.11}$
$(\#of)^{3.11} + (\#of\ dep)^{3.11}$
$(\#of)^{3.11} + (\#of\ dep)^{3.11}$
$+ (\#of)^{3.11} + (\#of\ dep)^{3.11}$
Offering complexity
$(\#of)^{3.11} + (\#of\ dep)^{3.11}$
$(\#of)^{3.11} + (\#of\ dep)^{3.11}$
$(\#of)^{3.11} + (\#of\ dep)^{3.11}$

(Source: Author's own)

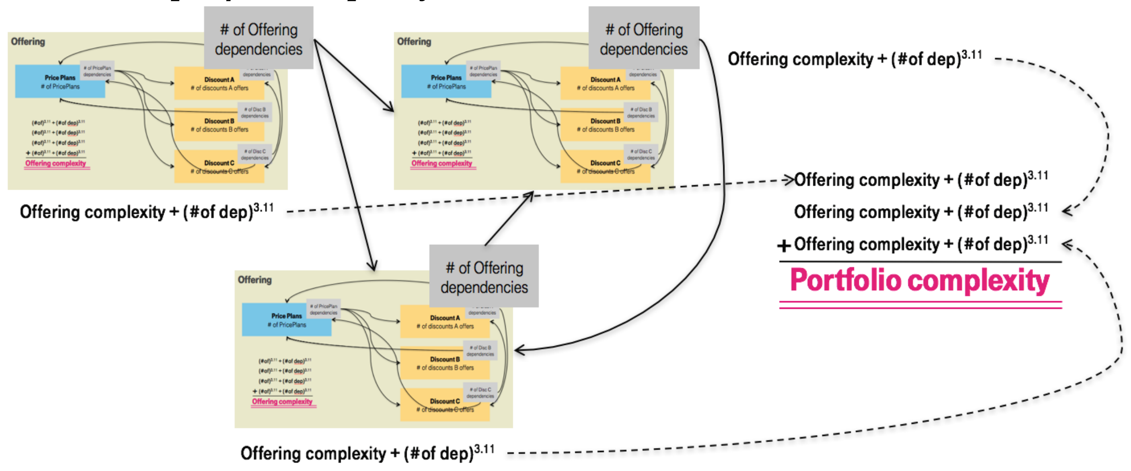
Product rules as dependencies

Before jumping up to the level of portfolio complexity calculation we should stop a short while to clean up the product related rules behind dependencies. Everyone who worked with order management/order capture knows the basic rule types controlling sell ability of products: preconditions, exclusions, move together and cardinality. The rules can be inter-product and environment-product types taking the following definition as a baseline: a rule contains the condition and the control as the two main building block. The product related rules has a product on the controls side anyway. Only residential customers (condition) get Tariff A (control); active Product A (condition) allows to buy Product (B). This two examples described two precondition rules, the first is environment-product and the second is inter-product type. Using this knowledge, we can define the boxes and the lines now. The boxes will contain substitute products, which may have kind of exclusion rules to each other, but that should not been calculated, since Sessions' method defined that dependencies inside a box are not counted in the complexity. The other set of inter-product rules must be handled on the control side, while the environment-product ones are on their product (notice, that their place also selected by the control side!). We reached the stage that we have a calculation of offerings and using the algorithms of defining the boxes and lines we are sure that their results are comparable.

Calculating the portfolio complexity

Portolfio complexity is calculated using the offering complexity as argument of the calculation equity (replacing $M^{3.11}$), and N will be number rules controlling the offering changes.

Figure 10.: Product portfolio complexity



(Source: Author's own)

Voila, we have the portfolio complexity index! The real difference is comparing to other calculations, that we used the result of a previous run instead of finding countable items to use for calculation.

What is the cost of complexity?

The complexity measurement approach declares, that TCO, the total cost of ownership is driven by the complexity. The double of SCU is the double of TCO. Since ratio of TCO components differ project by project it is not so simple to find what will be the effect of changing a complexity for a given project (Thomason, 2010). Anyway, TCO has two main portions: CAPEX and OPEX, the investments and the operational expenditures. In the case when the software or organisational environment is too complicated, CAPEX will increase. If the solution is not optimal for the requirements, processes are overcomplicated, OPEX will grow (Spacey, 2017). Hard to define the clear impact of the problems in the ecosystem. This is the reason of looking for the cost equivalent of SCU!

Cost of delays

All we agree that the project delays have costs, means losses. In general, the losses have two main types: the additional resources and the delayed incomes. The extra expenses of additional resources are still the simpler problem, while the specialities of the projects cause high deviation between their costs of additional resources. As a result of this high variation, no general correlation can be found, which can help to build awareness of this affect for the workers. The delaying incomes affects the net present value (NPV) and the internal rate of return (IRR) of a project – a suitable value of them is the condition of project start-up in project based organisation – can be calculated from the analysis of the expected cash-flow of a given project. As in the case of the costs, the incomes are too special for each project to produce a generic method to investigate their affects to the project losses (Csiszárik-Kocsir – Varga, 2016b, 2017).

It seems we reached a dead-end. If we would focus on individual project we did. We have to use statistical approach instead! If a company has relevant number of projects, then we can take statements of an „average” or “statistical” project, which will reflect to the organisation. Again, we need a good approach! The calculation approached described in the Price tag of project delays article (Nacsák, 2017), gives a way to have an approximate value for an organisation, to highlight the NPV erosion per month in the given environment. Beside the details, having a company calculates 36 months payback time, has 20% RRR, will lose around 3,5% of the NPV per each of the first six months of potential delay.

4. Hypothesises

Summary

Having CCE in hands we will have a useful tool, which helps to decide between multiple solution options replacing the currently used cost estimation approach with an exact calculation. It will then be possible to compare the cost of complexity difference with additional costs of other options like licences, support fees, cost of delays, etc. Determining CCE will help one more thing to do: planning simplification projects. Companies with large sophisticated enterprise architecture has a common problem: all they want to be simpler, but these activities cannot be designed with engineering precision, without having an exact measure what they can optimise to. Without the exact measure it is not possible to make proper decision between the different solution option. SCU and CCE will be the solution for this problem. Fighting with the complexity is nothing else that fighting with an ultimate rule of nature: the entropy. The experiences show, that randomly with no proper knowledge about the complexity no stable and on long term reliable solution by our everyday practice. In addition, we see the large number of perfect creatures, environments in the nature. The problem for us, that in corporate environments there is no chance to develop solutions steps by step through thousands of generations using evolution programming (Hillis, 1999 p142-144) - like nature does. On the other hand, if we cannot count on the forces of evolution, we have to have conscious design methodology or engineering-kind operation to reach the best possible results. It has infinitely small chance that a tornado sweeping through a junkyard might assemble an airplane, while building an airplane is not a single problem for engineers. Since engineers knows the way how to do it!

This jump may seem big, but there is a reason being here. Among the imperfect solutions described above, I remember quite a few exceptions that worked for an incomparably long time compared to other developments, surviving all attempts to replace them: these without exception, came out of the kind of the “quick and dirty factory”, as temporary solutions! So can there be a force in the architecture like the mysterious quantums in the Life of the edge (McFadden-Al-Khalili, 2016)? Perhaps an even more mysterious, currently unknown extra force, energy will then be released as the phenomenon articulated in Jeremy England’s DDA-theory (Wolchover, 2014), that creates stable nodes despite increasing entropy? I definitely don’t know the answer to that question. On the other hand, yes, we can achieve much better results as a “flight engineer” by supporting the great methodology found for the engineering work of architectural design with financial foundations. A small step in this direction is therefore to determine the cost equivalent of complexity.

References

1. Csiszárik-Kocsir, Á., Varga, J. (2016a): The value based analysis of the financial. The journal of macro trends in social science 2:1 pp. 89-100. 2016
2. Csiszárik-Kocsir, Á., Varga, J. (2016b): The Hungarian SMEs' Project Financing Practice – Results Based on a Primary Research. In: Marko, Kolakovic (szerk.) Proceedings of 2nd Business & Entrepreneurial Economics (BEE 2017) Conference Zágráb, Horvátország: University of Zagreb, Faculty of Economics and Business (2017) pp. 163-169., 7 p.
3. Csiszárik-Kocsir, Á., Varga, J. (2017): Financial knowledge, skills and investment practice in Hungary - results based on a primary research. Macrotheme Review: A multidisciplinary journal of global macro trends 6 : 4 pp. 10-20. (2017)
4. Glass, Robert L. (2003) Facts and Fallacies of Software Engineering. Boston, Pearson Education, ISBN 032 111 7425
5. McFadden, Johnjoe – Al-Khalili, Jim (2016) Life on the Edge: The Coming of Age of Quantum Biology. New York, Crown, ISBN 978 030 798 682 5
6. Hillis, Daniel (1999) Üzenet a kövön. Budapest, Vincze kiadó, 161.p ISBN 963 9192 27 9
7. Sessions, Roger (2008) Simple architectures for complex enterprises. Redmond, Microsoft Press (US), 256.p ISBN-13: 978 073 562 578 5
8. Taleb, Nassim Nicholas (2013): Antifragile - Things That Gain from Disorder. Penguin Books Ltd (UK), 544.p ISBN: 014 103 822 5
9. AEA US (2017) Telekom's live architecture knowledge base: case study presentation, Letöltve: https://atollgroup.eu/telekoms-live-architecture-knowledge-base-case-study-presentation/ (Utolsó letöltés: 2020.11.10)
10. Nacsák Tamás (2017) The price tag of project delays, Letöltve: https://www.architectarchers.com/en/news-feed/30-articles/141-how-much-you-loose-if-you-delay (Utolsó letöltés: 2020.11.10)
11. Session, Roger (2011) The Mathematics of IT Simplification, Letöltve: https://www.architectarchers.com/images/documents/MathOfITSimplification-103.pdf (Utolsó letöltés: 2020.11.10)
12. Spacey, John (2017) 11 Examples of Complexity Costs, Letöltve: https://simplicable.com/new/complexity-cost (Utolsó letöltés: 2020.11.10)
13. Thomason, Derek (2010) The Cost of Complexity, Letöltve: https://opexsociety.org/body-of-knowledge/the-cost-of-complexity/ (Utolsó letöltés: 2020.11.10)
14. Wolchover, Natalie (2014) A New Physics Theory of Life, Letöltve: https://www.quantamagazine.org/a-new-thermodynamics-theory-of-the-origin-of-life-20140122/ (Utolsó letöltés: 2020.11.10)