Chess and the art of Enterprise Architecture

Gerben Wierda

Free Excerpt

Making the right moves to manage Business–IT complexity

Published by R&A, The Netherlands

Typeset with TEX, using the mkii version of the ConTEXt macro package.

ISBN 978-90-819840-5-8

Contact info: *info@enterprisechess.com*
Web site: *http://enterprisechess.com*
Suggested hashtag: *#eachess*

The world is not $\mathbb{Q}$, it is $\mathbb{R}$

The most erroneous stories are those we think we know best — and therefore never scrutinize or question.
Stephen Jay Gould

Contents

In software, the chain isn't as strong as its weakest link; it's as weak as all the weak links multiplied together
Steve McConnell — Code Complete

Nothing is as dangerous in architecture as dealing with separated problems
Alvar Aalto

Prologue: Loosely Coupled Spaghetti

SUPPOSE WE have a system for the approval of loans. The business logic has been put into a Business Rule Engine (BRE) system*. Our system has business rules and for a certain class of prospective customers, an extra check is needed (thus say the business rules). This check involves sending a message to a credit rating institute and waiting for the result. Our system will in those cases use our Enterprise Service Bus (ESB) to send a message to a server (which runs a service for this function, hence Service Oriented Architecture or SOA) which then handles the interaction with the credit rating institute via a dedicated secure link. Since we only do this check in a small percentage of cases, the server handling the requests and replies to the credit institute is limited in size and our secure link is narrow as well.

The system complies with all the current popular architectural terms, is up and running, and all is well.

* If you are not technical and terms like 'application server', 'caught exceptions', or 'web proxy' in the fragment that follows, do not immediately get you started, do not worry: there is no need to know what I am talking about here in a technical sense. You can read this without understanding any of the technical references. Just skip over them, their specifics are not important to the story. They are real enough, though. You can trust me on that, or ask your engineers.

Now, a credit crisis comes along. Suddenly, the credit risk of prospects becomes an important factor. "Simple", says the business consultant, "we just change our business rule so all but the 1% wealthiest of our prospects are checked against the records of the credit rating institute". It is a simple change to make, and thanks to our Business Rule Engine, no programmers are needed. Management sighs its relief. The change is made, the result is tested, and the whole new business rule goes into production...

...and immediately grinds to a halt because neither our server handling the interface to the credit ratings institute, nor the connection to the credit ratings institute are capable of handling this amount of activity. The program on our application server creates many 'time out' exceptions because the credit ratings institute does not reply fast enough, and as a result our application server dies from memory problems as it tries to handle all the loggings as a result of the caught exceptions in the application server. Because the application server has died, the web proxy server starts to present an incomprehensible error screen to the users, which en masse call the help desk. All these calls to the help desk forces the help desk people to put up a taped answer, frustrating the users even more, while the infrastructure engineers scramble to dig into the setup to find the root cause of the incomprehensible error screen on the web proxy server.

Now, people may say: this change should have been stress tested and of course it should have. But as both Dijkstra[1] and McConnel[2] have written: tests cannot prove the absence of errors. Besides, not all situations can be made so that actual real-world tests are possible.

This story, however, is not about how this problem should have been caught. This story is about the fact that we have created — with all our state of the art tooling, our ESB, our SOA and our BRE — a setup where you make a small change in one place and the setup starts to go massively wrong in another. Such a problem with interdependencies first appeared in computer programs during the sixties of the previous century, when programming was new and those programs started to become sizable (and thus complex) for the first time. The programs turned out to be very *brittle*; they broke easily. For the software architectural problem behind it, the phrase 'spaghetti code' caught on. Since our modern state of the art setup exhibits the same sort of behavior, we must conclude that we have spaghetti again. Loosely coupled spaghetti, but spaghetti nonetheless.

It is not that we cannot solve this particular example problem, or that it is a particular good example, it is that — with all our object orientation, service orientation and business rule abstractions — we are back in a situation where the word spaghetti applies again, roughly half a century after the term was first coined as a derogative term for something that should be avoided.

And not only do we still have spaghetti, even some old simplistic (and ineffective) remedies are still being practiced. If the application server can't handle it, your engineers will propose to 'solve' the problem by adding more computing power and memory to your servers. This 'solution', many of us will recognize, and as a result we all also have the experience that, while hardware is supposed to become cheaper all the time, technical infrastructure for large setups is still very expensive. But scaling up this hardware or that connection may (temporarily) ‘solve’ a particular problem, like the one from our example, but it is essentially the same kind of ‘repair’ as was undertaken fifty years ago when a programmer increased the size of some array in a computer program to prevent that program from crashing. Such a change may prevent the crash for now, but it does not address the root cause, and the problem generally crops up again and again until you fix the real reason why memory is filling up.

There was already talk of a ‘software crisis’ more than forty years ago[*]. And in 1986, Fred Brooks, famous author of *The Mythical Man-Month*[†] wrote[‡]:

> Of all the monsters who fill the nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors. For these, one seeks bullets of silver that can magically lay them to rest.
>
> The familiar software project has something of this character (at least as seen by the non-technical manager), usually innocent and straightforward, but capable of becoming a monster of missed schedules, blown budgets, and flawed products. So we hear desperate cries for a silver bullet, something to make software costs drop as rapidly as computer hardware costs do.
>
> But, as we look to the horizon of a decade hence, we see no silver bullet. There is no single development, in either technology or management technique, which by itself promises even one order of magnitude improvement in productivity, in reliability, in simplicity.

[*] E.g. in Edsger Dijkstra’s ACM Turing Award lecture in 1972 [Dij72].

[†] [Bro78]

[‡] [Bro87]. And I must say, I’m a bit envious. The man can write. This is beautiful prose, which I wish I could have written myself.

Brooks further argued that the (then) new methods (like object-orientation) had been able to minimize the (programmer-made) 'avoidable complexity'*, but that the 'essential complexity' could not be solved by clever methods, it was just a fact of the landscape. There was, as Brooks wrote, "no silver bullet".

The belief that problems, such as the one from our example, should be at least partly solvable has remained alive, even if, from the mid-nineties on, both the avoidable and the essential and irreducible complexities of the IT field have largely been the expected and accepted backdrop against which our trade is performed. Nobody talks about a software crisis anymore, if alone because staying in crisis mode is not feasible for decennium after decennium. Skilled IT or enterprise architects already know this and will not be surprised by the above example of 'spaghetti', it has become 'normal'. We are just not used to calling it 'spaghetti' anymore.

Now, the example above serves two objects. First, as a simple to understand warning for organizations and their non-technical managers, that there still is no silver bullet. The warning is not unimportant, as the industry is still rife with silver bullets for sale, all unknown when Brooks wrote his article. ESBs, BREs, SOAs, Agile, Scrum, Cloud computing, and all kind of platforms are often still sold as such (and believed in as such). For instance, while a bit on their return, Business Rule Engines have been popular with management, because they promise a shift from programmer to the 'business consultant'*▸, and thus away from the need of

* Brooks actually calls this 'accidental complexity', but 'avoidable' is better because this results not only from chance but more often from lack of skills

(incomprehensible, brittle and expensive) IT skills. Recent additions to the 'silver bullet' set are, for instance, Cloud, Agile, and Scrum. All worthwhile methods and technologies, certainly, but no silver bullets.

The fact is, that using an ESB, a BRE, SOA, Cloud, and all the others does not by itself provide stability, robustness, and good and future-proof systems. Without some of them, we might indeed end up with systems dying in a gridlock of hard dependencies, true, but the price we pay is an increase of loosely coupled spaghetti-like dependencies. And IT is not only complex, it is also fundamentally brittle because in itself, as a digital environment, it lacks the flexibility of human-type compensating behavior*. These days — thanks to networking and especially the internet — these dependencies range across all kinds of levels: software, application servers, networks and infrastructure, and even organizations. In part, it is managing this complexity of dependencies that the field of enterprise architecture was originally invented for.

*◂ The current (2014) buzzword for this seems to be 'low code' development.

* This is even true if the software itself has been built with robustness in mind. Software may be robust under — for instance – bad data input, but that only means the software will not crash. It does not mean the software will still function as need be. At a business level, if software does not crash but also does not provide its intended service, the *business* will 'crash'. IT-centric ideas of robustness in fact ignore that it is the business that needs to be robust, and that requires more than just *technically stable* IT environments. It needs *functionally* stable IT environments, and that is something that digital technology fundamentally cannot deliver for reasons that are outside the scope of this book.

The consequence of problems like our ‘loosely coupled spaghetti’, and essential complexity in general, are the difficulty and high cost — in time and money, and in frustration — of *change*[*] and *control*[†] in any organization that has a complex Business-IT landscape.

The issue at hand

Before computers were hooked up in all sorts of networks, writing good programs for them was already difficult, as illustrated by the quote from Brooks above. It was clear to many in the field that there were no silver bullets, and that better design — i.e. *system architecture* — was to be the solution.

When computers seriously started to get connected via networks, the problem became even more complex. System architecture alone was not enough, as there were many systems in play. The logical next design phrase was *enterprise architecture*. Thus a discipline was born.

Since then, networking has grown many orders of magnitude — from the small Local Area Networks, or LANs, of the eighties, via the narrow links of the initial internet and other Wide Area Networks (or WANs) of the nineties to the high speed broadband internet and the SaaS and Cloud solutions of today. During that period of rapid growth of connected systems, the discipline of enterprise architecture has been around as a way to fight the greatly increased complexities of the Business-IT world. And not just IT has become more

[*] E.g. large IT projects.
[†] E.g. risk and security.

complex, business has become more complex too, in a large part because IT enables it to become more complex.[3]. The complexity has grown in a close tango of Business and IT.

The question is: can enterprise architecture solve the problem of essential complexity or is it another silver bullet itself? Should enterprise architects maintain the position that the essential complexity is solvable at all? Because by maintaining that it is possible to solve the essential complexity probably, while not solving it at all, enterprise architects are positioning themselves constantly as amateurs and failures. And if complexity can be managed by enterprise architecture, how?

Sadly, we must conclude that enterprise architecture has not been a success story. True, the field is established, there are methods, frameworks, conferences, books, specialists, departments, and so forth, all very busy with enterprise architecture. But ask most organizations that do something with enterprise architecture, and you will not find many that are very satisfied with the *results*, even if they sometimes will say that they are satisfied with those that perform that function.

This book is an *argumentative* book. I will argue that much of the 'best practices' of the current enterprise architecture discipline have fundamental flaws, and that even performing them to perfection will not bring what they are supposed to bring.

First, I will roughly describe the essence of current practices, without going into detail of all the various frameworks and approaches. If you are acquainted with enterprise architecture, much of this will be familiar territory, so it might feel that I am taking too much time to get to the point.

There are however two important reasons for including this bit of ground work. First, this book is written for a wider audience; the reader should not need much prior knowledge. Second, we architects often differ subtly in our definitions and assumptions, so I need to share my framework of what enterprise architecture is, before discussing its flaws.[4]

Next, I will explain why our 'best practice' approaches most likely *cannot* work as expected, and also why we use them nonetheless. I'll draw on fundamental psychological, biological and philosophical subjects. But don't worry, I'll not go into these subjects very deep, even if these subjects do merit such attention. This is, by design, neither a heavy nor a complicated book.

My personal favourite philosopher, Ludwig Wittgenstein, by many seen as the most important analytic philosopher of the previous century, has once said that sometimes, looking at a tough philosophical question is like having entered a room without any other exit than the one you came in by. In such a situation, he argued, going to extreme lengths to try to answer the question is like trying to get out of that room without exits, except the door you entered the room by. You can always leave by that door of course, or in other words, you can leave the question for what it is, it is apparently not a question that can be sensibly answered. Enterprise architecture today is like someone who has entered that room and doesn't know that his approach is doomed. He needs to retrace his steps and go elsewhere. I will do that too. I will present a way out of the locked room, a way to get out of the corner enterprise architecture has painted itself in. Having tried that alternative approach over the last years have convinced me that it is possible, though not easy.

A personal note

This book grew out of my personal development as an enterprise architect. Before ending up as enterprise architect, I experienced many different roles in and around IT: software engineer and team leader for a large chemical company, software designer and IT infrastructure manager for an AI-related company, postmaster in the beginning years of the internet, a bit of sales, owner of my own small software development firm, civil servant working on government strategy for science & technology, manager of a department where a lot of high-tech research & development was going on, information strategist, and finally back to the more technical world of Business-IT, or enterprise, architecture.

When I returned to (enterprise) architecture I constantly felt unhappy about the state of affairs of that discipline. And I suspected that the remedies offered — often as 'best practice'[*] — were unlikely to work. They reminded me of the failed simple ideas that flourished at the start of the Cognitive Science/AI developments of the sixties and seventies of the previous century, that had been so eloquently unmasked by Hubert Dreyfus in his wonderful book *What Computers Can't Do*[†].

My own ideas on how it *might* work slowly matured. When I had the chance to implement these ideas, the vague ideas met their first clash with reality[‡]. They turned out to work, but I also experienced the problems and bottlenecks of

[*] One of the most to be distrusted phrases in our industry.

[†] [Dre92]

[‡] Speaking of which, in case you are curious: the statement on the opening page is explained in the endnotes appendix[5].

trying to do that — and I experienced the necessary failures to sharpen my understanding.

Somewhere during that period, when I was trying to explain in a positive manner (instead of critiqueing the basic problems with existing enterprise architecture approaches) to my environment what my ideas were about and how they differed from 'best practice', I thought of the basic analogy of enterprise architecture and playing chess. I have been using that analogy very effectively since, to explain enterprise architecture, why it fails so often, why many of its standard assumptions are misguided, and how I think it should be executed.

As mentioned above, this is by design not a heavy volume, and I have severely limited references to everything ever written about enterprise architecture that I know of[*]. It is small, because — besides addressing my peers — I also want non-architects such as general management to understand *why* fighting complexity and unpredictability is so difficult, and *what* can be done about it. A bit of practical information[‡] that may be useful if you want to practice enterprise architecture along the lines of the ideas in this book has been relegated to the appendices, so it will not get in the way of a quick read but are available when required.

I would like to thank Guido van Kilsdonk, Joost Melsen, Jean-Baptiste Sarrodie, and Wil Scheijvens for the feedback received on various drafts of this book. I also want to thank

---

[*] For instance, you will not find a reference to the TOGAF, FEAF, DYA, or other enterprise architecture frameworks in the Bibliography, even though they are mentioned here and there in a sideline[†].

[‡] Background information, templates, . . .

[†] This is *not* a book about (existing) enterprise architecture frameworks.

Joost Melsen specifically for valuable discussions during the last couple of years — when the ideas now written down in this book were put into practice. His critical remarks on my rough ideas made those ideas substantially better, and his exceptional qualities as an enterprise architect were indispensable. He is the best enterprise architect I know.

I am also endebted to a couple of authors. A few of the most important ones for me are philosophers: Ludwig Wittgenstein, P.M.S. Hacker and Hurbert Dreyfus. Let nobody tell you that analytic philosophy has no practical value; the sort that is closely related to the real world is. I also would like to mention the editors of the *New Scientist* magazine for creating an excellent gateway to the wider world of science & technology*.

I would like to apologize to Renée, Renske and Mark Douwe for what they had to suffer while I was writing this (and during the years I was — sometimes painfully — being taught by exposure of my ideas to the real world).

Finally, I'd like to thank you for purchasing and reading this little book. I trust you will enjoy it. If you want to discuss it with me, use the email address *info@enterprisechess.com*. More discussion and articles may appear on the book's web site: *http://enterprisechess.com*. On Twitter, I suggest using the hashtag *#eachess*.

Heerlen, December 2014,
Gerben Wierda

* And no, I'm not trying to kiss their most illustrious bottoms.

Enterprise Architecture is the discipline of managing the complexities of the Business-IT landscape. It has been around since the 1980's, when for the first time computers were connected in networks, and the already serious (and unsolved) problem of the complexity of computer programs for relatively simple business needs turned into the huge problem of large networks of them in complex business landscapes. In spite of many 'best practices' and 'frameworks' that have been introduced, Enterprise Architecture is not a great success. After thirty years, we still have the same problems. Chaos is still everywhere. Projects still fail far too often.

In this book, (hidden) assumptions behind the existing approaches to enterprise architecture are challenged, and a more realistic perspective that helps us battle the complexities and unpredictabilities of today's Business-IT landscapes is described. Practical suggestions about enterprise architecture governance and products, based on real-world experience with the described approach, complete the book.

From general management to IT professionals, everyone who is confronted with the problem of managing Business-IT landscapes can profit from the insights this book offers. No specialist prior knowledge is required.

Gerben Wierda is author of *Mastering ArchiMate*, and was, amongst other things, Lead Architect of the Judiciary in The Netherlands, Lead Architect of APG Asset Management, and is now Team Coordinator Architecture & Design at APG. He holds an M.Sc in Physics from the University of Groningen and an MBA from RSM Erasmus, Rotterdam.

ISBN 978-90-819840-5-8

9 789081 984058 >

Free Excerpt

Published by R&A
2015