Supporting domain experts to become citizen developers

Using mapmaking: themes, abstraction and goal based thinking

By Karl Jeffery and Dimitris Lyras

Sept 3 2021

Published by Software for Domain Experts
39-41 North Road, London, N7 9DP, UK
www.softwarefordomainexperts.com

SOFTWARE FOR DOMAIN EXPERTS

Contact Karl Jeffery on jeffery@d-e-j.com

Contents

Why support domain experts to become citizen developers

What this book is about

Wouldn’t it be great if people with specialist needs for software could have more of a role in developing it?

This book will explain why this is a good idea, why it is an important idea, and what you can do to make it happen.

We’ll show what the obstacles are and provide some ‘mental’ tools which can help overcome them – themes, abstraction and scripted thinking. We’ll give some examples about how it can work, and where it can lead to.

We use the term ‘domain expert’ for someone who has specialist expertise which is relevant to a certain area of an organisation’s activities. That is an enormous range of people – examples can be senior management, people who operate and fix equipment, people who plan cleaning rosters, people who diagnose patients or make marketing plans.

This book is about these domain experts becoming citizen developers, developing and improving the software that they use.

How organisations work

Before we delve into the arguments about software, here’s a note about what organisations do in society and how they function.

The effectiveness of our organisations affects the quality of almost everything in our modern lives.

Education, healthcare, defence, physical security, social security, regulation, employment. Management of food, transport, clothes, money.

Our ability to combat climate change depends on effective organisations. Only an effective organisation can deliver us the products we expect, demand and are used to, but make less emissions in the process.

We have organisations because they can do what we can do by ourselves but more cheaply. We can survive without organisations. We can defend ourselves, grow food, transport ourselves, make clothes, and create a financial system, just as we did in prehistoric times.

But organisations make it all happen more efficiently and effectively, by doing it on a bigger scale, and breaking down the tasks into small pieces which individual people can get a deep understanding of.

Effective organisations have people in specific roles monitoring, learning, and making decisions about some specialist aspect of it. We can call these people domain experts. They are the people who understand some key area of an organisation and would like to continually improve their understanding.

If we are going to improve how the organisation runs, it will be hard to do it by searching for better people to be the domain experts. We’re stuck with the people we have, like you and me.

But there is potential for these domain experts to get better tools to understand what is going on in their specific part of the organisation, continually learn more about how it is working, and make decisions. With better digital technology.

We could get better digital technology from large software companies. But we'll show in this book why this is unlikely to happen, at least directly, because making specialist tools does not usually fit the big technology business models.

The best way to get better digital technology is probably going to be if the people who use it – the domain experts – are able to get as close as possible to the development process or do part of the development themselves.

So, this book is about supporting domain experts to become citizen developers.

The term "citizen developer"

Citizen developer is a term we first heard in 2021. It builds on efforts which have been going on for a decade or more to develop 'low code' or 'no code' software tools. These can be used to build software without coding expertise.

The literal definition of a citizen developer is someone, who is not a professional software developer, who is developing software.

With this definition, we have a concept which sounds exciting to company managers. They have to pay large amounts of money on professional software developers. Plus they see plenty of other overheads involved in buying or commissioning software. The meetings where people try to explain what they want, the IT department finding software companies, the software company team leaders, product managers and salespeople, trying to persuade the company IT department that they have what is needed.

They think, it would save a lot of money if we could manage without all these people, and let the people who use software build whatever software they want themselves.

But the idea of citizen developers is very scary to actual developers. Not because they think their jobs could be replaced by ‘citizens’. They know their work is harder than it may look. Plumbers do not generally worry about their jobs being replaced by citizens either.

But professional developers know how getting software working, while itself very difficult, is only a small part of the total job. You need something which can be robust and secure for many years. There needs to be clarity and transparency about how it works, so someone else can understand it and add to it later. You need to be careful that your changes to the software do not cause problems somewhere else, by understanding how different systems depend on each other. So even if a citizen developer could develop something which works for them, it may not work for the organisation.

So the debate about citizen development in 2021 seems to end up with a very low level of ambition. When actual projects for citizen developers are discussed, they are fairly simple projects which don’t need to integrate with anything else.

For example a tool to connect a database to a user interface with a little logic. To calculate how much to pay someone, to store data about an asset, to make a calendar base schedule, to store data about performance, to generate quotes, to keep track of tasks.

This book aims to raise the ambition level. Overall, we want to see the use of software in organisations reaching its full potential. We want to see specialist domain experts able to use specialist digital tools, but without their companies paying enormous sums for specialist software developers they may never meet face to face.

We are not going to talk about citizen developers being able to develop or improve all software that the company uses. But we imagine that their ability to change things is something which varies, just as their ability to change anything else in the organisation varies.

In some areas they can develop their own software, in other areas they can improve it or define improvements, with other areas the domain experts can request or suggest improvements which may be considered by a group of people, and may not be considered.

Just like the organisation itself works, finding a balance between ensuring consistency and continuity, and allowing continuous development.

We want to combat what we see as two big weaknesses in the conventional model of software delivery to organisations. These constrain what digital technology can do in the big picture.

Firstly, the usual software product business model rewards companies which are able to provide the same product for multiple customers. It does not reward companies which make specialist tools. Software companies sell their products by making big promises about how they can solve all their clients' problems, without acknowledging that their clients' problems are all different.

If the organisation has thousands of users doing the same thing, and the same as people in other organisations, the usual software business model is fine. But there are plenty of organisations which are not like this. And the solution is not to have more smaller software companies doing the same as big software companies but on a smaller scale. This business model does not scale down.

The second weakness in the conventional software business model is that domain experts do not have any easy means of correcting faults in their software logic, or even suggesting improvements to it. So they end up with software which follows a different model to the one they use in their heads.

The authors’ experience in shipping

The ideas in this book were developed out of the authors’ work with digital technology in the commercial maritime (shipping) industry.

This is an industry which puts people under very high risks in personal safety, and has seen many serious accidents in the past, thankfully fewer in the past two decades, but the risks remain.
The industry has been made safer not through software, but through a mixture of systems and human expertise.

People working in shipping come across different situations every day and use their domain expertise to exercise judgement in how to minimise risks.

Maritime software could do a great deal more to help these people understand their situations, such as from informing them, at the right moment, about experiences of others who have been in the same position in the past.

But because all situations are different, it would take a great deal of expertise to develop such software. Because activities in the industry are so complex and diverse, it would be very hard or expensive to make software which did this perfectly first time. So this software would need to be designed in such a way that it could be continually improved from the experience of people using it.

But we don’t yet have ways for domain experts to improve their own software, and the industry does not have huge budgets for developing perfect-first-time software. So we have limits on how good the software can be.

Example from the travel industry

Here's another example of where software does not achieve its potential, in an industry you may be familiar with even if you do not work in it - travel, such as hotels and airlines.

Have you ever had a situation with a hotel or airline, where both you and the staff member assisting you wanted to make a change to a plan or to your account, but the software would not allow it, or made it very difficult? Or making the change was theoretically possible with the organisation’s software, but was very hard to do, and beyond the capability of the staff member you were talking to? Probably because the software defined the situation according to its internal logical steps, which are very different to how you would handle the situation in the real world?

Some examples could be changing a booking for one type of hotel room to another, because one room option was not available when you originally made the booking, but it is available now. Or extending the length of stay, but finding that your existing key card will only work for the length of stay in the original booking. Difficulties adding more family members to a booking. Complexities with refunds or credits.

The hotel or airline is trying to push you towards a chatbot or junior member of staff to resolve the problem. But the only person with the skills to achieve the change using the software is a more senior manager, who is hard to reach.

In a pre-computer era, the changes could be made in an instant. Yes, you can change your room, extend your stay, add people to a booking, and with pen and paper I work out that you pay or are refunded this amount, here it is in cash. Perhaps nothing would need to even be written down, just a revised verbal agreement between the customer and the hotel manager, and perhaps a cash transaction.

Another example of problems caused by software is a hotel midnight fire alarm caused by someone drunk and smoking in their room. In a pre-software era, the issue could be identified and resolved very quickly. Today we would have automatic fire alarms forcing all guests out of their rooms, and automatically calling the fire brigade. Meanwhile the night staff of the hotel do not know how to use the software to identify the room, switch off the alarm, and tell the fire brigade, and in the stress of dealing with this, nobody thinks to do the most important thing, inform the guests they can try and go back to sleep.

When hotel staff are talking to customers, they may diplomatically call these situations "computer says no". Or say that the computer restrictions are acting in everybody's best interests, to maintain security or avoid something bad happening. But what is actually happening is a computer causing problems, obstructing the staff member for doing what they want to do.

And what would resolve the problem is if the software could be improved so it can actually follow the logic which the domain expert uses in their mind (the staff member handling the problem). It would be click, click, click, done.

You - a Person who Supports Citizen Developers

We propose a new organisational role of the person who enables domain experts to become citizen developers.

Let’s say, that person is you.

To be good in this role, while you’ll need to be confident and comfortable in the world of digital technology, you don’t necessarily need a background as a software developer. A background as a software developer may actually be unhelpful, if it encourages you to look more at the code rather than the real world which the code works in.

It will help if you have good analytical and logical skills to understand the basics of how computer systems work and their logic.

It will help if you have a good capability and interest in understanding organisations and how they work.

It will help if you are good at communicating and explaining, both orally and written. You can take in a large amount of information, abstract in your mind what is important, and present that to someone else. Perhaps you have written essays or have considered being a journalist.

It will help if you have good empathetic and listening skills, because you’ll need to talk and learn from a lot of people. It may help if you have worked in a pub or restaurant, and understand what a ‘service mindset’ is.

Humility will help a great deal, since you will be in worlds you will initially have very little understanding of, your initial ideas about how they work will need to be revised many times.

You’ll need to be resilient, practical, and able to keep at something even though people may try to dissuade you.

And most of all it will help if you are the sort of person who likes to understand what is going on behind what you can actually see, so you can better understand it. And someone who can think for yourself, to understand a situation you haven't seen before.

This sounds like a big demand. But if you're in your, say, 20s, educated to degree level, shouldn't all of this be part of your basic skillset?

A bumpy pathway to citizen development

Technology becomes our servant

This shift to citizen development could be accompanied by a shift in how technology is treated in organisations, where it becomes our servant.

Fitting the organisation around technology, where the technology is really our master, is something which has happened going back to the 1970s, when we had the first software systems, such as for accounts or library books.

Today, software has such a big role in accounting and libraries that the role of staff can be largely about entering things into digital systems, such as scanning a library book someone wants to take out, or updating accounting software about transactions. And much of people's focus ends up making the software do what they want, such as when a library system refuses to accept a book, or there is a complex accounting transaction to try to enter into the software.

In organisations today, what is called 'work' can be a miserable existence of endlessly logging onto different packages. These are software packages which might have been never designed for our specific organisation, and do not integrate together.

Rather than the software being fitted to how we work, we have needed to adapt how we do our work into something this software can handle.

In a world of domain experts doing citizen development, this all changes. Librarians go back to their core role of helping people find books they want or would like, and encouraging interest in books. Accountants are managing the company's accounts. They have fixed all the problems with the software so it interoperates perfectly and invisibly with their work, and it doesn't distract anybody's focus.

Connected with this change of technology becoming our servant, we put AI in its place. Much of the excitement about AI in the past decade has been driven by the idea that AI tools can do our work for us, or even do it better. For people who work in organisations, this means AI becoming the master.

The idea of AI as a servant is something different, and probably more useful. Servant AI would help the domain expert get a better understanding of what is happening in the domain, such as by processing large amounts of data to identify trends or anomalies which a domain expert can look at more carefully.

Technology people sometimes use the term 'high level' to mean closer to the organisation and people, and 'low level' to mean closer to how the computer thinks and works.

Traditional software development has been about getting from the 'high level' world of people and organisations to the 'low level' world of code. The experts are the programmers who learn how to convey what an organisation does in a way a computer can understand.

With citizen development and low code, software development is about going in the opposite direction, taking the 'low level' of the code to the 'high level' of the organisation.

A world where software is made more efficiently

In this future technology world, software would be provided far more efficiently, via a services and platforms model.

The usual software business model works like this. Software companies are seeking to develop individual products which can meet the needs of thousands of potential clients, so they can be both very general or have large amounts of functionality which will never be used.

Software companies are trying to develop new functionality for their products so they can compete with competitors to increase sales. In doing so, they complicate their own products, and develop more functionality which may never be used.

Software companies try to recover the money spent developing this functionality by finding ways to ‘lock in’ their customers, making it difficult to switch software, resisting efforts to make it easy to integrate.

Meanwhile domain experts in our organisations often have a good idea about what software tools would help them, often through frustration with what they have been given. They may have good ideas about what needs improving in the tools they work with. But their ideas are never implemented.

In the future technology world, we can see software companies evolve from being a vendor of a product, to a provider of a service which is within their sphere of expertise, since the software company may also employ domain experts.

With this change, we are moving to a much closer collaboration between people who use software and people who develop it. We are removing all the communication layers and barriers which normally exist between ‘users’ and ‘developers’ - including IT departments who choose and buy software, salespeople who sell it, requirements engineers and business analysts, product managers and developer team leaders.

Who will support this?

Getting to a world of citizen developers will be bumpy. It is not in everybody’s immediate interests.

The people easiest to convince of this new way of working will probably be domain experts themselves, who will be happy to adopt a way of working which makes their work easier. But they often have the least say in how new software is developed or chosen.

Big software companies may quietly oppose this shift. They want to keep their products entrenched in organisations and have a big interest in the status quo. Members of company IT departments, with people building careers on their expertise with specific products, may also favour the status quo.

Not all digital technology people will oppose this shift though. There are already plenty of digital technology companies orientated around providing services rather than products, such as the “software as a service” companies. And many people in the digital technology world enjoy thinking creatively, working out new ways to achieve goals, and ‘disrupting’ the status quo.

Your role

Your role is to develops skills to help an organisation and its domain experts get to this new world.

You may be quite young and inexperienced. In this case you also have an uphill task because people well into their careers often don’t take advice easily from someone in their 20s.

But the organisation’s domain experts will come to appreciate you when they see how you can help them get on top of their digital technology and make it work for them.

They will appreciate you more if you are a good listener, learner, and explainer, you are able to adapt your plans when you need to, you have a sense of practical solutions, and you want to help people to do better work.

This could be a good career option for you, because it is somewhere where you can generate enormous value in organisations.

As someone early in your working life, you may feel your existing options are doing work which people older are not willing to do because it is so arduous, or doing work which requires less skill and so can be paid at lower rates than older people are willing to work for. Or doing work in domains where older people do not have skills, such as software development. So this provides you with a new alternative.

To do this, you’re going to need to sit down with the domain experts and understand what they do, their model of the world. You’re going to need to understand the digital systems that already exist. Then you’re going to work out how the situation can be improved and how you can help people to get there.

In a later chapter, this book will outline three ‘mental tools’ which may help you – the ideas of identifying themes, abstracting, and script-based thinking.

In the process of doing this, you’ll support our organisations to move in other directions which you may approve of, where they are driven by experienced individuals making good decisions, rather than by bureaucratic process.

You'll support non authoritarian countries to compete much better with authoritarian ones in both economy and quality of life.

You will help keep any digital systems transparent rather than mysterious, make their data easily accessible and usable, and make it much easier to maintain and demonstrate cybersecurity.

Why this is inevitable

Consider that the trend towards more citizen development is surely inevitable.

Just as some people want to design their own houses, choose their own friends and clothes, as soon as they have the capability to do it.

Digital technology development has favoured a centralised approach up to now. That's because the business case in making code favours 'one to many'. Code it once, sell it to hundreds of thousands. Manage things centrally.

Many aspects of society were also centrally run, at points in the past. That pleased the kings, barons, and other leaders we had. But as soon as it became possible for people to run their own lives, that's what they did.

Your obstacle - people who think organisations are about script following

The biggest obstacle you may face is that many organisations, perhaps most, are not designed to support the creativity this needs. They are designed to create and enforce processes, or scripts.

That’s how the people who run them think is the best way to make them profitable or effective. And many people who work in them agree, and feel most comfortable in this environment.

If an organisation truly achieves effectiveness from doing the same task over and over again, and getting better and better at it, with little variation in both demands for what it does and its ability to deliver, then it does not need any of this.

But what’s more likely is that the organisation needs both script-based thinkers and abstraction based thinkers, and it needs to make room for both.

The organisation needs people who can think for themselves outside a script if it needs to deal with unusual situations. Shortages of staff, shortages of supply, changes to demand. Regulatory, societal, and environmental changes. New competitors. Or just something happening which hasn’t been seen before.

In these situations, the organisation cannot rely on its script followers. It needs to rely on someone’s judgement. And for that person to make judgement, they need the best possible understanding of what is going on. Supported by digital technology.

Understanding domain experts

What do domain experts do?

In order to support domain experts to become citizen developers, the first step is to understand what domain experts do, what they need, what the computer systems providing them need to do, why the conventional digital technology delivery system doesn't provide it, and how the situation can be improved. This is what this chapter will explore.

Domain experts, in this book's definition, are the people who make the judgements or decisions in every organisation, based on their understanding of how things work, and the information they have.

The senior management are domain experts, but they are not the only ones. The people who co-ordinate, schedule, monitor, advise, fix, buy, recruit, train, operate equipment, look after people and things, are domain experts.

As examples, domain experts need to decide what their priorities are at any time, what is most worthwhile spending their time on today. They may be making schedules and plans, involving customers, staff, assets, or tasks, delivering certain outputs most effectively and at the necessary time. They may be making purchases, and judgements about whether this specific purchase is the most appropriate choice.

They may need to assess the competence of other people they are working with, in their team or under their management. They have to assess risks, including fraud, safety, cybersecurity, compliance with regulations. They need to determine if there is a pattern of events happening which may give insight into what is really going on.

These work processes all involve situation awareness and decision making. They need to know what is happening, and determine whether they are making the right choice.

When domain experts do their work well, the organisation functions well, doing whatever its role in society is.

Having well-functioning organisations working towards a goal is the pathway to fixing big problems in society. Such as improving cybersecurity or making decisions to minimise climate impact, supporting people's education, or giving support to people in difficulty. All of these involve the right people having the right situation awareness at the right time.

To achieve the societal goals, the societal goal needs to be accepted as the organisation's goal. That is outside the scope of this book, and probably involves the right incentives being set by government, investors, or customers. But once this is done, and it is increasingly being done, the next challenge is the thousands of decisions people in the company make every day, so they can best achieve these goals.

Another way to understand how many domain experts work, and how digital technology can help them, is to see them as detectives. They are putting together information from multiple sources, together with their own understanding, to try to work out what is happening, or what happened. Or if something is different to what they would normally expect, or if they can see how something can be better.

To be a good detective, it is helpful to have transparency, which could be defined as the ability to get answers to questions that arise in their minds. Such as when a detective working on a crime realises it would be helpful to be able to ask someone a certain question and get a reliable answer. Where they are using digital systems, they want them to be clear and flexible enough to provide this.

In their heads, they build models about how they think things work. For example: I change this, and that happens. This cause drives this effect. When I see this indication, it means a certain something is going wrong. When I have to decide between two options, this is how I choose the best or least risky one.

If we ask, do decision makers in our organisations have the best possible understanding of a situation based on the data that's available, including the large amounts of new data the organisation has started gathering since it became much easier to do? The answer is probably no.

The things domain experts want to find out and decide on have an enormous range. Let's take an example of a manager of a coffee shop which is part of a chain. Much of the decision-making is being made by head office, such as about presentation of the brand, the products, the visual design.

But the manager is still left with many decisions to make, including how to evaluate staff, how much supplies to order, how many staff members to schedule at different times, how to handle unhappy customers or problems with staff, what to do when equipment is broken, or crimes are committed.

Note that domain experts do not necessary all follow the same model in their minds, even two domain experts doing the same task. As in the example above, two different coffee shop managers may make different hiring decisions. So when supporting domain experts in their decision making, such as with digital technology, a tool which works for one person won't necessarily work for another in the same role.

When domain experts use software

When domain experts use software, the ideal is that the digital tools deliver this situation awareness, they should fit with the domain expert's mental models. The domain expert is able to understand how the software works and its logic, and have some mechanism for the software to be improved according to their suggestions or requests.

That sounds complicated. It may sound simpler as an example. Consider an aeroplane pilot. The pilot uses digital tools to get situation awareness of what is happening, such as the plane's position and altitude. The tools are designed to fit the pilot's mental models. When the pilot sees a need to increase speed, turn, or prepare to land, the tools to do that are immediately available and intuitive to use.

The pilot needs some sense of how the instruments work, and how their data is processed. A number of well-known accidents have occurred because sensors, and the processing of the data, were giving a pilot wrong information and the pilot was not able to diagnose this. We can imagine many accidents were avoided because the pilot was able to diagnose that the information being received is due to faulty sensors, not something bad happening outside the aeroplane.

While we would not expect a pilot to alter the software in an aeroplane, you would expect some feedback mechanism where a pilot can inform manufacturers of when the software gave inaccurate or misleading information, where the software failed to diagnose that it was being fed data from a faulty sensor, or where its logic performed in a way which was different to what was expected according to the pilot's mental models, making it less intuitive to use.

Another way to explain how domain experts want to use software is to say, good software is software we don't notice at all. For example, you may have dealt with many different parcel courier websites when tracking parcel deliveries. You may have noticed that some are so intuitive you barely notice them, as you find out where your parcel is, make sure you are home when it arrives, or give instructions for a redelivery. Other parcel websites seem to give you many unnecessary barriers, such as requiring registrations, and do not give you much useful information at the end of it.

Just like with any other tool we use in our lives, if we don't have to think about the tool as we use it, that means it is a good tool.

Not distracted by software

Building on this, if domain experts are to focus on understanding a situation and make the best decisions, they should not have their focus pulled away by software. Such as putting effort into figuring out how the software works, what its internal logic is.

You may have had experience working out a computer system's logic when trying different approaches to buying something online to see how you get the best price. But this is not the sort of thing we want our organisational domain experts to be doing.

This problem is compounded for someone working in cybersecurity. The best way to understand hacking attempts is through a real world lense. Why did someone want to do this, how was it made possible or easy to do it, or easy enough to justify the expected rewards? How did they obtain the password?

How does the bank I have an account with ensure no-one else can access my funds? How does this sensor prevent someone from tampering with it, or how would I detect if that was happening? Why would someone want to tamper with it?

But too often, cybersecurity attention goes entirely on the digital systems themselves. Cybersecurity people often have an IT background, and IT people often find it easier to think about digital systems than people, and so their focus easily goes there. So we get more and more digital controls, such as demands for more complex passwords. When people start writing the passwords down or sharing them to make it possible to do their work, this approach may have reached its limits.

Being able to improve the logic

In the organisational software world, our digital technology will typically have logic in it which is designed to make our lives easier.

There may be logic to identify what it thinks are duplicates, such as when someone enters an invoice into an accounting software which already has a record of an invoice with the same number. Or someone enters a name in a contact management system when someone with the same name is already there. The software may warn us when this happens and allow us to check, or it may automatically remove what it thinks is a duplicate.

Or there may be logic to determine whether the plan we are making is workable, such as giving us a warning if someone needs to be in two different places at once, or actually rejecting our plan.

This logic can be a great help when it works, but very painful when it does not work. Think of the feeling you have when Microsoft Word autocorrects something which was correct to begin with, such as the company name IHS autocorrected to HIS. And when your work involves safety risks or large amounts of money, the pain is greater.

The de-duplication logic may be classifying two items as duplicates when in the real world they do look similar but also have some small difference the computer system disregards. Such as two very similar parts from the same supplier. Or the logic classifies two items as unique, when they have different ID numbers, but are actually the same thing, such as parts made by different suppliers which can be used interchangeably.

But it could be cost effective to make better methods for domain experts to get this logic corrected, updated, or aligned with their preferences. This may actually be the best business case for citizen development.

The domain expert is not actually ‘developing’ software. But we are not seeing the world with developers on one side of a barrier and ‘users’ on the other. And computer code is itself logic. There is no barrier between the roles of saying how software logic should work, saying how code logic should work, and making code. It is fair to say, the domain expert is a citizen developer.

What domain experts don’t know

Domain experts would not normally be experts in digital technology unless they are digital experts. So they would not normally know what is possible for software to provide, or what they could demand.

This is an additional complexity in your role of enabling domain experts to become citizen developers.

The organisations which employ domain experts expect them to solve problems, and know what tools they should use to get there, since they are the domain experts, after all. So if the domain experts are not demanding better software, the organisation is unlikely to demand it on their behalf.

So a big part of your role will be educating, helping domain experts to better understand how digital tools could help them more, and knowing what a good tool looks like, so they can ask for it. The later chapters of this book should give you some ideas how to do that.

Why “big tech” offers limited support to domain experts

To go further into a point this book made previously, big technology companies typically want to make products which many different industries can use. Their ‘product’ business model gives them most profitability when they can build something once and sell it thousands or millions of times.

People often use the terminology vertical and horizontal to describe this situation, where vertical means a niched industry, and horizontal means something which can work across multiple industries. The narrower the niche, the less attractive it is to big technology companies. Conversely, technology companies make big profits when they can make the same product for multiple industries, which could be called horizontal. Such as general office tools, software for accounting, software for sales management, human resources, and in some cases software for maintenance and purchasing.

It would suit big technology companies if their customers could be standardised in their needs. Since the real world is not very standardised, this is not going to happen. But there’s an area of conflict in the middle, where a company may be persuaded to adopt a generic tool which proves to be unsuitable for their specific needs, or if a company finds that it is preferable to use a lower cost, more generic but also more robust tool with thousands of customers, than a higher cost, specific tool, which has few customers and also more bugs.

But to deliver the sort of organisational improvements described in the beginning of this book, more generic software is probably not the way to go.

A specialist industry has specific data which is worth collecting, or needs checking, or passing on to someone specific, and it needs specific knowledge to know how to do it. This drives the need for specialist software, and specialists to be involved in making it, or able to improve or correct it.

A specialist industry like maritime has its own internal language which domain experts use to explain clearly to others what they mean. It might be useful for digital technology to be able to understand this language when it is in written form. But that would also require specialist expertise, something most big tech companies would not be interested in developing themselves.

Big tech companies may be interested in making algorithms for how thousands of identical pieces of equipment work, like the output of a large manufacturer. But they are unlikely to be interested in helping a company with thousands of different pieces of equipment develop algorithms, such as a maritime company, or any industrial plant.

None of this will be made clear in the sales process. Big tech companies employ salespeople with a brief to show how their tools can solve any business or social problem. This can be a promise very far from the truth.

That outlines your challenge

Now we have outlined your challenge in supporting domain experts to become citizen developers.

While there are many potential benefits, the people who will directly benefit the most, the domain experts, may not understand it, or be ready to demand it. The organisation which employs them will be unlikely to provide the facility if the domain experts themselves are not demanding it.

And the big technology industry, which supplies most digital technology, and has much influence on organisational technology overall, might not see this approach as being in their interests.

But there could be a great pathway open for you, if you go into this problem with humility, open mindedness, a willingness to engage with people and understand what they do, if you have a strong desire for finding workable solutions to problems, one step at a time. If you can connect things together, particularly the computer and digital worlds. You could become a very useful person to your organisation and the domain experts in it.

For this to be a good pathway for you, you will need to be ready to understand that if people in powerful positions tell you something won't work, it may mean that it just doesn't fit with their vision of how the world works, not that it doesn't work at all. But they may not want to hear it.

Themes, abstractions, and scripted thinking

Introduction

In this chapter, we'll look at three 'mental tools' of themes, abstractions, and scripted thinking, which might help you understand where you need to go and what is stopping you from getting there.

Firstly, we'll look at themes. People think in themes, computers don't. But in traversing the 'digital' domain to the 'conceptual' (or human mind) domain, we're looking for themes. This is how people understand what is happening in their world, and understand how their digital technology works.

Secondly, we'll look at abstraction, the process of moving from the highly granular world of digital systems into a picture someone can use to understand what is happening.

Everything in computer systems is highly granular – data from sensors, data in databases, enormous volumes of e-mails, and whatever else. But information needs to be 'manageable', or much more abstracted, for people to work with it. You can think of it like a process of mapmaking – reducing large amounts of data into something smaller and which you can easily absorb. Such as a theme.

Thirdly we'll look at scripted thinking. This is the belief of people who may oppose you, that we can get where we need to go by following pre-defined steps. There are no pre-defined steps in enabling domain experts to become citizen developers, abstracting, or developing themes. You need to be able to think for yourself.

The alternative to scripted thinking, which you need to help domain experts become citizen developers, is goal based thinking – a mental orientation around the goal. The best pathway to get there may change, so you have to be always ready to drop your script.

The combination of goal based thinking, abstracting and finding themes is something which could be called 'mapmaking' – working out the pathway to take your organisation, or your task, where it needs to go, based on what you have available, and working out how digital technology can help you on that path.

Themes

We could define a theme as a group of things together.

If the theme is 'summer', that implies a group of things we expect, such as weather, clothing, activities.

As human beings, our minds make the world easier to understand using themes. We are evolved to think in themes.

Just about anything in the natural world can be seen using themes at any level. Personal relationships, nature, geography, evolution, changes in time.

The reason we are talking about themes here is that computers do not think in this way. Computers can work with data about the world with enormous resolution. If for some reason we have a summer but where the weather is colder, more like winter, that's no problem for a computer to grasp.

The world of organisational digital technology has often been about pushing people to see the world the computer sees it, without any themes. We sometimes have to work with tools in a very granular way, entering detailed passwords, entering account transactions at a journal level, reviewing database data directly.

A computer system developed to use themes would have no cybersecurity challenges. You use your laptop in a completely different way to anybody else, with different patterns for software applications, activities, times of day. In other word, your use of a laptop could be a theme. But to a computer, the only thing which differentiates you from a hacker is something very granular, a knowledge of a complex login password.

Themes are important in the working world, and good organisational software might be built to support these themes, rather than push us to work without them. Domain experts doing citizen development could help us get there.

Consider that an ice cream van operator knows that the amount of business tomorrow depends on what the weather will be, and orders supplies accordingly. So their ordering software could start by checking tomorrow's weather forecast.

Imagine explaining a crime to a police officer. The police officer is mentally filing your report into different categories based on their experience. Is this domestic violence, robbery, related to homelessness, mental illness? Is this ‘petty crime’ or something bigger? Does this threaten people’s sense of safety on the street? Is this something really serious like a child abduction? Is the person telling the story credible? These are all themes.

Using themes makes it much easier for the police officer to be effective, quickly categorising a problem, and identifying how much resources to allocate to it.

A doctor may look for certain themes when assessing a patient, such as the person’s age, or indications about their diet and lifestyle. But their computer system will just give us data from sensors or forms.

Other examples of where themes would be used are solicitors making an initial review of a case, a schoolteacher determining what would be most useful for a class, an engineer assessing a mechanical problem.

As human beings, we seek to reach a point where what we observe fits a theme we know. At that point, we feel comfortable that we understand a situation well enough to make decisions.

Themes are not necessarily right or wrong, and different people can have different sets of themes while working in the same environment. Like two investors which use different rules of thumb. What matters is that they work for the people concerned, which they presumably do, since if they often led us in the wrong direction, it would eventually be clear, and we would stop using them.

Politicians try to create themes, such as a theme for the cause of a drop in people’s living standards, which sounds believable. We could define an election as a choice about which politician’s theme we deem most credible.

Cybersecurity can be hard because initially there are no themes. But until we have distilled what we see into themes, we haven't understood what is going on.

Examples of common cybersecurity themes relating to hacking a website could be hackers trying to obtain people's passwords via a hackable website then trying the same password to get into their e-mail; a move towards multifactor access to services, such as a text message sent to a phone in addition to a password; attempts to steal SIM cards. Themes relating to phishing could be making sure staff are aware of the threat, using alternative means to mitigate the risk of a hacker obtaining a password, and improved e-mail screening systems.

Themes in procurement

As a more specific example of how we might use a theme in organisational digital technology, here are some common themes used by people who have a role in organisational procurement, to understand what is happening.

A common theme is for a supplier to charge less for the initial contract, and charge more for services after sales. Just as PC printers are sold cheaply but their cartridges are expensive.

Another theme is when owners try to bypass these extra charges by going to an alternative supplier, also as we see on printer cartridges, there are alternative suppliers selling for about half the price.

Alternative suppliers thrive if they can match their products to what the equipment needs, although the manufacturers may provide limited information about how their products work, to make this harder.

Perhaps the provider of the equipment, such as the printer, will go to a separate supplier to make the parts (the cartridges) – in which case it will ask them to sign a contract preventing them from selling their parts to customers directly unbranded.

It is not implausible for purchasing software to incorporate these themes.

Themes and solving big challenges

Here are some examples of how we use themes in solving big problems, including climate change, cybersecurity, and fair cash distribution.

A theme for climate is a switch to electric cars. It bundles many complex questions together, such as the emissions made in generating electricity and battery manufacturing. But by and large experts agree that the move is better for the environment because it allows a release from oil-based fuels and a path to renewable power for transport. So, working with the theme makes life easier.

In cybersecurity - a simplifying theme when solving the challenge in the big picture is to try to assess the overall attitude a company has to cybersecurity, using a deeper layer of themes of what we would expect good and bad to look like. This can be easier for an auditor to determine, rather than looking at the system from bottom up, in a granular way, inspecting the digital systems themselves.

For the question of fairness of cash distribution in society, people relate this to the theme of fairness in education. It is not directly linked to fair cash distribution, but is indirectly related, and easier to influence.

When themes meet computers

The world of people-with-themes meets the worlds of computers in two different places.

One is when we are building computer software which helps someone to understand the world. Could the software help us categorise what we are seeing using the themes we already use? What sort of crime is this, what sort of problem are we facing here?

The other is when people are trying to work with or understand digital systems - it is easier if we can understand the digital system using themes. For example, if it is a website, we could have themes relating to how the website was built and by what sort of company, if its construction is based on web standards, how old it is, do I need to see the code to understand how it works, and if so, how easy is it to understand the code.

Why people want themes and computers don't

The importance of themes to people, and not to computers, can be explained better when you consider how computers and minds work differently.

A computer can handle enormous data sets, but with fairly narrow processing - either rigid calculation models or logic, or the potential of machine learning, but still only learning some narrow thing. It can only approach human capability of doing a broader task, like driving, if that task can be broken up into thousands of sub tasks which can all be programmed narrowly.

A computer itself can only work at high granularity, following very precise instructions about how to do every little thing. We have programming languages which reduce the granularity to some degree, but they don't take away the need for precision, and can’t, because a computer can’t guess what we want it to do.

Meanwhile a person can handle only small data sets - such as the 8 or so phone numbers we can all remember. But we can handle information with a level of richness, variability, or depth a computer cannot approach. We evolved to be able to live in tribes of 100 people, understanding them all as individuals, and having a relationship with all of them, including negotiating power, building relationships, and in some cases controlling others. Plus understanding what we needed to survive with the help or otherwise of our tribe.

Themes and driving, people vs machines

Here’s a way to explain how important themes are to the success or failure of digital systems in the real world – the story of autonomous car technology development.

The concept of themes helps us understand how people and computers drive cars in completely different ways.

When people are driving, we use themes, such as the weather condition and what driving style is appropriate, what we think someone else is doing based on what we know about them, what the regulations say. Each theme is like downloading a book into our mind and invoking details we are not consciously aware of.

The computer on the other hand is not using themes but trying to create a detailed picture of what is around the car - other vehicles, people, objects, and where they are moving to, and what road signs say.

The weakness of the computer, and of autonomous car technology development to date (up to 2021), is that this detailed picture approach only works if you can understand every necessary detail, which means that a computer has been programmed to understand it. If something happens outside this scope, the system fails.

Meanwhile a theme-based approach can easily be extended to something new. And a trip in a car will often involve an event which has not been seen before and so a computer has not been programmed to understand.

Take an example of a traffic light which seems to be stuck on red, a vehicle not moving in the road, a person standing in the road. While these would all challenge an autonomous car, a human car driver would immediately search a deeper mental directory of themes to figure out what to do.

Am I sure the traffic light is broken or is it just programmed for a long wait? is there a vehicle detector which can be woken up, is the street quiet enough that crossing under a red light is safe? Is the vehicle broken down or likely to move in a few minutes, can it be driven around safely? Is the person in the road mentally ill, had an accident, or protesting, and what does that mean about whether it is safe to drive around them?

Themes in software building

To get all these themes working in software, it is useful if the theme is carried through as deeply in the software as possible. To use software development language, we want the theme to 'persist'.

In a typical software development process, the discussion between developers and domain experts, which will include identifying themes, may only happen in the beginning as part of 'requirements gathering'. But then the software developers are left by themselves to build what they think the domain experts want. The domain experts' themes can easily be lost.

For example, a medical expert may say that they would like software which can categorise the symptoms they see into the themes representing cause A or cause B. There will be lots of knowledge behind this theme making which the doctor uses, which won't necessarily come up in the initial discussion with software developers.

The software developer may make software which can work with available data from sensors on the patient's body, and write logic which can try to allocate this to a theme. But without the medical expert being closely involved in the software development process, the logic could easily end up wrong. And without a means for the doctor to subsequently update the software, the software would be useless.

And in real life, the doctor is looking at multiple data points, including from speaking to the patient. The doctor may be invoking subconscious knowledge from experience. It is possible to build software which incorporates this, but not with software developers left to themselves.

Using abstraction to find themes

Our second mental tool we discuss in this section is abstraction. The mental process of generating themes could be called 'abstraction', because you are coming up with a simplified version of reality, but which is easier to work with.

Domain experts often don't do the abstraction to create the themes themselves. Much of it is passed along from one domain expert to another, or taught when people do their formal training. "Here's how you know that the driving conditions are wintery enough to change your driving style."

When one domain expert shares with another what they think is important, they are also sharing methods for how to abstract what they see into themes.

Abstraction is very hard mental work. To be good at abstracting you might want to adopt some of the techniques which creative people use, such as making some time and space clear from distractions in order to do it, and being aware that people with other thinking styles may try to get in your way. And note that even creative professionals cannot usually be creative for 8 hours a day.

Abstracting is not a skill which people today develop or are used to developing. Like a muscle which is rarely used. As a younger person you might have a more developed abstraction skill than people many times older, because it comes out in play, or when doing anything creative, or even having conversations with a group of people in the pub.

To describe being funny in a very boring way, it often involves abstracting, coming up with an interesting theme to explain what is going on in an unexpected way.

You may be good at abstracting if you have high levels of curiosity, you want to understand what is happening, and see it in different ways, and from different perspectives, and find ways to make it all fit together.

If you are the sort of person who likes to observe different management styles, such as autocratic vs collaborative, and seeing where they work and don't work, you may be this sort of person.

Another way to explain abstraction is to think about what a journalist does. The journalist researches a story by getting deeply absorbed in what is happening, through conversations with people and reading material about it. Then they produce a short written article which is itself an abstraction of what is in their minds.

Anybody teaching anything will often use abstractions, so they can connect the gap between their own in-depth knowledge of something, and their student’s lack of knowledge about it.

Any musician connects the details in their music about what should be played, with the audience’s need for a picture which makes sense as an abstraction.

Abstractions are made for the purpose of goals

The reason we abstract is to achieve or understand goals. All of this happens intuitively, but we are breaking it down here to better understand how it should happen in the context of digital technology.

For example, how to achieve our own goals, how to understand other people's goals, how to support people in their goals.

As human beings, we can usually mentally grasp goals easily, although we may vary in our capacity to associate granular data with goals. For example, some people are better than others in abstracting granular data about Covid cases, deaths and vaccination levels, to recognise that getting a vaccination should be their goal.

The simplest form of abstraction to explain is working out how we can better achieve our own goals. A geographical map maker could make many choices as to what is and isn't included - it could include wildlife, history, population, businesses, geology, for example. But the purpose of the map is normally to help us to get somewhere, so there is a priority on roads and place names.

Abstracting to understand other people's goals is also something we do intuitively, but not something any computer could do. Just as we could easily abstract to understand the purpose of a room, from knowing a few details about what it is it.

Consider this story. Roger complains to Bob that he can’t get restaurants to cook hamburgers rare enough for him. Bob replies that he can get his barber to cut his hair short enough.

The abstraction, which Bob and you understand, perhaps intuitively, is that Roger is talking about the problem achieving a goal of persuading a service provider to do something unusual. Bob replies with a story of his own problem with this goal. This is nothing to do with hamburgers or barbers, something a computer would never understand.

Supporting people in their goals using abstraction is something we might do with computer software. Consider how an accounting software package might, on being opened, tell the user, a small business owner, how many bills are becoming due to pay, how much cash is in the bank, how much money is coming in, any urgent tasks, and what the cashflows look like over coming weeks. These are things a small business owner would have wanted to know since we had businesses, money and bills, and their models for these things will already be ingrained in their minds. But when the data is all in a software system, providing this is an abstraction from the granular data in the system.

If the data relates to what is going on in an organisation, then by making abstractions, we create a more general picture, which may be useful to someone when relating something from their memory about a similar case which happend in the past. In a similar way, a journalist may tell a story about something specific, but relate it to general goals, which a reader may relate to, such as taking care of children, or seeking a better job.

Making abstractions is scary

Making abstractions is scary because it is difficult - and because other people may not agree with the abstractions we make. We are used to working with others at certain abstraction levels which feel comfortable. So, creating new abstractions within an organisation is very hard.

The more intellectual energy / deeper engagement which is involved in making an abstraction the more fear it invokes. So doing good abstractions becomes a direct fight against our own fear. Perhaps this is the biggest challenge. Perhaps there are ways an organisation can be structured to support or push people to get beyond their fears in making abstractions.

Sometimes we pretend to make abstractions because we think it makes us look clever or interesting, but what we are actually doing is borrowing abstractions which have already been created and tested by others, but we get better than others at spotting and stealing them. Such as when someone shares a clever sounding political opinion about what is causing what is happening, which was originally created by someone else.

Agreeing on an abstraction

Often, not all the domain experts in a company have the same abstractions.

In supporting citizen developers, you may need to want to get a consensus between people about how the system works - or have a system which can work with differing views. This would enable us to have a single digital tool everyone can use.

Getting a consensus on what is going on can be hard work but plenty of people know how to do it, such as anyone who has ever chaired a meeting.

If you are not seeing agreement you may need to increase the abstraction level, until you have a core goal which everyone agrees on. For example, people in a community meeting may have different views about what is most urgent, but a consensus can be achieved by abstracting to a core goal everyone agrees on, such as safety, enjoyment, and outside play, then having a more concrete discussion about which methods would best achieve this.

The right abstraction is not always obvious. It may only be achieved using creativity, guesswork and finding out if the guessed abstraction resonates with other people. Such as when a comedian tests out new material on a live audience.

Abstractions and digital integration

The link with digital technology is that we use abstraction to stride the gap between the granular data in a computer system, and the abstracted understanding which people have.

This will be easy enough to do when the input and output purposes are related, such as entering data about a school class attendance which you later use to determine the attendance of the class.

But this gets much more complex when the data is being used in different ways, as it increasingly is in our organisations, such as if the attendance data is then used to compare schools.

The data, once collected, is just items in a database. But if it is only seen in this way, rather than a class register of attendance, then a lot of value is lost.

Let's say we have data about people, containing their first name, last name, passport number, date of birth. But we have lost our record of where this data came from, how reliable it is, and whether there may be any risk of duplicates. Is this data of any value now?

Here's another example. Consider what data you might have about your kitchen. Your kitchen has chairs, a table, a cooker, brown walls, it is in a building which is 40 years old.

We can get plenty of data about all of these things. The power of the cooker hobs, the height of the chairs, the ages and colours of everything.

If we combine this granular data, it would be very hard to make sense of. The electric hob uses 13 Amps, the seat of the chair is 40cm high, the toaster is 5 years old, we paid £180 on a bread machine. In our database we would have to keep track of what exactly the measurement is. Perhaps in future we have new chairs, a sandwich toaster, and a gas hob, and we are still trying to use the data.

If this was organisational data, we may want to use it in ways that we can't predict when we gather it. Such as, we want to assess how much the whole kitchen is worth, find out if we have room for another piece of equipment, find out if another person can use the kitchen, see if it is suitable for a disabled or obese person.

But this is what people often try to do with data integration projects, they just put the data together. It is easy to do technically, like pasting two columns into a spreadsheet, but not much use in the real world.

So, we need a better way to combine this data together than just throwing it together into one database.

Perhaps the best way to do it is to integrate it together similar to how the real world integrates together. In the real world, it makes sense to integrate chairs into a kitchen. This fits with a goal we all understand, of making food and eating dinner, with the people who we share the house with, at the same time. We all understand the goals of a kitchen.

It doesn't make sense to integrate data of chair height being 40cm and cooker hob being 13 Amp. There is no sense of any goal here. But if we were talking just about chairs, or just about cookers, then there would.

To apply this idea to a more complex real world organisational problem, let's consider a police detective has data files from a mobile phone, a report from a witness statement, a past history of a suspect. A detective would not, we imagine, consider integrating this into a data lake.

The best way to use it may be to keep it forever in terms of its abstraction, a mobile phone data, a witness statement, a past history of a suspect. These are all part of a detective's work, and when expressed like this, we understand how they support the detective's goals. If we compiled all of these data formats into a 'data lake' it would be harder to see how the individual data components help with the goals, and we might also have lost our track of how they fit together.

Here's another example of the value of integrating data in the abstract. As a car driver, you might find it very frustrating to encounter multiple temporary traffic lights because people are doing road works on a short drive you regularly take. You think, why are they digging up the same piece of road over and over again? Why are the temporary traffic lights so badly timed? Isn't this city meant to be one of the most digitally advanced in the world?

There is data about all of these things, but it is all in separate systems. This can include systems the water company uses for planning their pipeline maintenance projects which may involve roadworks, the system where water and communications companies apply to the council for the work they want to do, systems for programming temporary traffic lights, and Google Maps data about the current congestion levels.

If you asked a software programmer to integrate all this data together, they would assume you mean at a data level, which would be virtually impossible, and probably not a project anyone would ever attempt, because all of the data has a completely different context, and collecting each of the data sets has a different goal.

But if the data can be integrated together at an abstracted level, the people working with it can understand what is going on as easily as someone could read the paragraph above describing the situation. And once they have understood it, they can make steps to improve it, such as knowing which traffic light to reprogram, or which road maintenance program to reschedule.

Abstraction to connect data to insight

In supporting citizen development, you may be in a position of looking for better ways to connect the data the company already has with the understanding its domain experts need.

Company data could include all kinds of archives or ‘legacy data’, real time data such as generated from sensors, and something in between, such as data generated by other software applications, or data entered into a form.

You need to find ways that this data can be used to work out answers to questions like where the company is, what is happening, what to do next.

This is itself an abstraction task, trying to determine what useful themes could be determined from the data, and then finding out which of them would be useful to the domain experts you are working with.

One of the obstacles might be that you don't know about all the processing which has been done on the data. It means that people cannot comfortably work with data because they don't know what it has been through.

For example, if there is any 'edge computing' – that is, processing on data from sensors as it is gathered, before entering the data into a corporate system, then the logic of that edge computing needs to be very clear. Examples of this could include controls on a bank's website for verifying what someone wants to do with their online account, or a sensor's cybersecurity controls which stop someone changing the clock.

Abstraction and assessing technology

Could domain experts, and the people who support them, get better at assessing technology, with better abstraction skills? Even technology which has not yet been made?

At the moment, much of technology development is very expensive trial and error. Investors fund a number of companies in the hope that one will become a hit. Developers aim to build a 'minimum viable product' as fast as possible to determine if they are getting somewhere, which appears to do something useful but may not be the most useful possible thing.

Could we achieve the same result, at much less cost by using abstraction? We can use abstraction to understand what the goals are, what digital technology would best serve these goals, and whether the digital technology you are presented with can deliver this.

Scripted thinking

Our third ‘mental tool’ we address in this section is scripted thinking.

Scripted thinking is where people are following existing procedures, themes, and abstractions, and they don't have the need, capability, confidence or inclination to find new ones.

There’s nothing wrong with scripted thinkers, but if you’re inventing new themes and abstractions you need to be able to spot them, because they may not be able to understand what you are doing, and they won’t be able to do it themselves.

There can be a lot of scripted thinking mindset in organisations. It is a mindset which makes sense if you see your organisation as something which maintains the status quo and delivers goods according to a specific process. But it is the wrong mindset if you need to change or evolve anything.

Script based thinkers are focussed on developing the script. They think the goal is obvious, the challenge is building in the steps to get there with increasing granularity, and then following them, or ensuring other people follow them. Scripted thinkers can learn new steps quite easily, including new details between the steps.

The opposite of script based thinkers we could define as goal based thinkers, people who focus on where we need to go and are happy to continually work out the best way to get there, even if it involves dumping a script they have followed for years.

Script based thinking does not incorporate re-assessment of the world. Abstraction thinking is re-assessing all the time.

Scripted thinking can work in certain sectors where no abstracted thinking is required – such as a government organisation to enforce certain rules. Any suggestion of a change in approach can be dismissed completely.

But even here, it is very helpful if the people implementing the rules understand them and their purpose, so they can better interpret whether or not someone is in compliance, such as when they see something which breaks the letter of the rules but fits with their intention. Such as when a train passenger pays for a ticket in advance which is only valid on a certain day, but mistakenly selects the wrong day on the website.

Scripted thinking can be easier for people, it is also a mindset we can all default to when we are tired. It makes it easier for us to feel less anxious about our position in the organisation, since, after all, we are following the processes we are supposed to.

Sometimes scripted thinking people can be aggressive to others, because they both feel threatened by people who are seeking new ways to achieve the goals. They feel sure that scripted thinking is the right path, because that's the only type of thinking they feel they know how to do.

Although goal orientated people, when they are tired, may find it easier just to focus on what is needed to do to achieve the goal, and leave out all the scripted steps.

Business environments need both scripted and abstraction thinkers. Abstraction thinkers can be working out better ways to do things, spotting problems emerging or risks, or working in more inter-personal roles. Scripted thinkers can be managing finances and ensuring the company follows its processes. But, for no obvious reason, scripted thinkers tend to dominate.

If a group of people having a meeting gets too script-based, then they can lose focus on the bigger picture completely and just focus on improving the scripts. You may be familiar with this. For example, a meeting of a group of people working together on a document, who are having arguments about tiny points of grammar, and have forgotten what the document is actually saying or what it is for.

Script-based thinkers cannot understand or evaluate someone else’s script, if it is different, because to do that requires abstracted thinking. So instead, we can get a non-constructive fight over whose script 'dominates' - and the loser gets the tough task of trying to follow someone else's script.

Abstraction based thinkers can collaborate naturally, because you can abstract anyone's experience to the point where it agrees with anyone else's.

Script based thinkers cannot do well in a competitive environment where their existing script proves not to be a winning method. That would involve abstraction to the level of seeing whether there is a better way to achieve the goal, and maybe something they are doing is not relevant to the task, is creating inefficiency, and can be omitted.

Scripted thinkers in IT

In your work supporting domain experts to become citizen developers, if one of the biggest obstacles is script-based thinkers, it won’t help that you may come across many of them in the IT world.

Bear in mind, a computer itself is a script following machine. The role of many IT people is trying to make software run on computers, and trying to explain to other people how to work with the software, which also means following scripts.

They may typically resolve problems by thinking carefully about the script the computer is following, and where the activities do not fit this script. Did you not enter something, update something, load something, the way you were supposed to?

Using technology often requires complex script following. To travel by Uber requires you to carry a mobile device, load its apps, have credit cards, and follow the process for booking and paying for a trip, which all adds up to quite a complex series of scripts. No abstraction thinking required to use Uber.

This is not to say that all digital people are script followers. Many digital people are very creative. The people who conceive of new possibilities, set strategy, design and create products, motivate staff, understand the competitive landscape, do sales and marketing, and speak in public about the possibilities.

And it is not very desirable to have programmers who are purely scripted thinkers. Just as it is not desirable to have people operating heavy equipment who are scripted thinkers. You need people who can immediately spot where their scripts don’t fit the needs of the moment, and find a new approach accordingly.

Some examples from real domains

Introduction

Let’s try to illustrate some of the points in this book with examples from real domains. This chapter will look at domains which the authors work in and know well – maritime, cybersecurity, decarbonisation, call centres and competency management systems. The ideas in this book arose out of the authors’ work in these domains.

It is important to have some understanding of how these domains work to work out how domain experts might be helped to do their jobs better. Although as a note, fairly in-depth understanding of a domain is needed to have a good understanding of where digital technology can do more.

Getting in-depth understanding of a domain is hard work, and takes a lot of time and words. If you do not work in that domain, it does not generally make for interesting reading. So this chapter describes activities at a very abstracted level.

Maritime

The maritime industry, operating big ships, is full of domain expertise. Just about everybody has a role of managing something. This includes people at all levels and departments working onboard ships, and working in the office.

Nearly everybody is a domain expert, needing the best possible situation awareness, and all these domain experts could become citizen developers, because digital tools would really help them.

There is a lot of variability in the work, with ships constantly being sent to different places, carrying different cargoes or types of passengers, dealing with regulatory change, and also changes in the weather. So while there are a lot of processes, it demands what people call common sense, the ability to use judgement to understand what to do in a certain situation, where there aren’t written instructions, or scripts, available to you.

The industry’s activities are divided into commercial and technical operations. ‘Commercial’ means making decisions about which ships to buy and when to sell them. Also working with customers (cargo owners), negotiating charters (sort of vessel lease agreements), then planning vessels to carry specific cargoes, their route and speed.

‘Technical’ means actually running the vessels. That includes the crew onboard ships who navigate the vessels, do maintenance, and ensure safety, and the office staff who supervise the crew, resolve problems, plan maintenance, purchasing, and ensure overall technical integrity of the ships.

On the commercial side, the biggest factor which drives success in ship owning has been described as being able to spot changes in the market before competitors do – so you buy ships and sell them at the right time. This is fundamentally a human judgment issue, but many sources of information can feed it.

Then there are many decisions involved in the process of chartering a vessel, including agreeing on a rate, predicting how much it will cost to operate the voyage, and also keeping on top of demands from customers.

On the technical side, people would benefit from better situation awareness to know about the competency of crew they are about to hire, the condition of equipment onboard the vessel, any problems emerging with the vessel, emerging safety risks, violation of safety procedures, how much emissions of CO2 the ship is making and if they can be reduced, and much more.

Decarbonisation of shipping is a particularly important challenge of the 2020s. It involves gathering information about the emissions the vessel is currently making, and understanding if there are ways that the emissions can be reduced. This can be through operational changes (routing, which generators are in operation, how tanks are being cleaned), and more longer-term changes, like adapting a vessel to switch to different fuels, or other equipment investments.

There are many different ways to gather data from vessels, plot what is happening, make carbon scores, and understand the impact of changes such as different equipment and speeds. All of this has potential for citizen developers to improve their digital technology.

Cybersecurity

Cybersecurity is one of the biggest challenges of the 2020s. And solving it comes down to situation awareness in multiple levels.

If we refer to a commonly used cybersecurity framework or abstraction, it reduces cybersecurity to the themes of "identify, protect, detect, respond and recover." All of these involve situation awareness.

"Identifying" is about understanding what you need to protect, with perhaps some information assets needing more protection than others, just as you might put your jewellery in a safe. Different computer systems need different sorts of protection, for example corporate networks or online games for children.

"Protecting" is about knowing what the appropriate safeguards are, looking at your weakest areas, or places where you have most at risk. Weak areas may be phishing e-mails which your employees and systems are not good at detecting, sensor devices which may still use their default passwords, or unpatched PCs, or people with a lack of competence.

"Detecting" is about having awareness of what hackers may be doing. This can include understanding the various types of hack. It could include paedophiles trying to contact children via an online game, criminals trying to make money with ransomware or banking fraud, teenagers messaging around, or government attacks on what they consider adversaries.

"Responding" is about awareness of the best ways to handle an attack.

"Recover" might be about knowing how to restore a back-up or remove viruses.

One of the weaknesses with many cybersecurity approaches is that people take a purely technical approach, which can mean a script-based approach, and hackers develop skills of manipulating this. No-one would take a purely script-based approach to physical security, and the weakness of such an approach would be obvious. An approach to cybersecurity based around implementing a certain product can have the same weakness.

In the real world, the needs of every company for identify, protect, detect, respond and recover are different, and need awareness of different things. So you can see the value in enabling domain experts to build or configure their own tools.

Decarbonisation

Citizen development is particularly suited for decarbonisation projects, because there is a lot of new situation awareness which is needed.

Solving the decarbonisation challenge could be distilled to the question of finding ways to do what we need to do, but making less emission while doing it.

So having awareness of the carbon emissions associated with any choice, at the point of deciding, would be very helpful.

This can only be achieved by having good data about the carbon emissions which are being made due to choices you have already made, such as the choice of machinery, route, fuel use, or whatever else. Also having awareness of whether a choice with lower carbon emissions will affect financial performance in a means which may affect the organisation's financial sustainability.

In future we can expect more carbon pricing, which will change the calculation of how a certain choice relating to emissions will also affect finances. There are many financial aspects to be considered when making a choice, based around the costs and expected returns, and carbon prices will be added to this.

To understand carbon emissions about our choices, we need to gather and integrate information from multiple different sources, emissions from products bought, fuel we combust ourselves, perhaps leaks. Every company's needs are different, so it would be very helpful if the domain experts in the organisation could have a close involvement in the building and updating of such a system.

Call centres

With so much effort to enable people to do what they need online, we could say that people only need to use a call centre when they have a problem which is too complex to be solved with information on a website, or the website is too hard for a customer to use to find it. So that is already something of a failure in digital systems.

Call centres need many employees, and often have high turnover. So, there is not always much scope for training.

Software tools have been developed which can provide situation awareness to call centre staff, giving them recommendations about why someone is calling, and the best way to resolve the problems.

Software tools have also been developed to support management of call centre staff. Making tools which can count call length, who hung up first, whether the outcome was satisfactory, are easy enough to build, and commonplace. But perhaps they are not very useful, and make the working environment more unpleasant.

More useful would be software which can help track which issues people are struggling with most, any trends with more calls about a certain topic suddenly showing up. Also perhaps predicting which telephone calls will be most difficult, so they are spread out between staff, or perhaps diverted to more experienced people.

A call centre manager would have the domain expertise to know how this could be built. But would their expertise get incorporated into software?

Competency management systems

Competency management systems is a technical sounding term for something all teachers and organisation leaders need. They need to understand what competencies people in their class or team have achieved, where they have weaknesses, and what would be best at fixing those.

They also need to understand which of the multiple e-learning tools available is best at delivering what they need. Some systems are geared more to memorisation, some more to practise, and not everything can be taught using e-learning.

They also need an understanding of the best assessment systems. Digital tools can easily assess factual knowledge if people can express it using a multiple choice, but most useful knowledge is not easily shared in this way.

We could say that competency management systems are in their infancy, if we imagine a mature competency management system being one which would help both manager and student / staff member know exactly where they stand and where they need to get to.

It sounds possible to build using digital technology, but would be very specialist, and would need a great deal of customisation, aligning the computer's assessment of competence with what counts as competence in the real world.

Connecting citizen development with AI

To many people, AI is the most exciting thing happening in digital technology, and it would be unusual, and hard to understand, why someone would write a technology book without mentioning it.

Citizen development is not a completely separate world to the world of AI. But we need to think of AI differently to how it is usually done if we are to bring these worlds together.

A common understanding of the purpose of AI is to make machines think and do work. This is not the world we are working in here. This book is about supporting domain experts, not to replace them.

But AI can do a great deal to help domain experts. For example, it can simplify large data volumes and draw out the main themes from them, to make them easier for a domain expert to work with.

AI systems are great at spotting anomalies in big data sets, finding clusters in data, and trends and correlations. This is all extremely useful for people. But you need a domain expert to work out if the patterns could have a real-world cause. If not, maybe they are not a useful pattern.

For this to work, the domain expert needs a good understanding of how the AI is actually working, and its logic. Or at least, the domain expert probably will ask for this after being let down by the AI system once or twice and losing trust in it.

Domain experts can master skills in AI themselves. This has been seen in the oil and gas industry, with people working with equipment and subsurface data.

It is getting progressively easier to work with AI without having advanced data science or algorithm programming skills, with new tools being developed. But we are far from being able to program AI without a basic understanding of the algorithms.

The process of working with domain experts and finding themes can lead to domain experts being better able to express what they want AI to do, and someone else being able to deliver it for them. Programming AI tools is expensive, so it is useful if this goal, and the value from achieving it, and business case for investing in it, can be clearly expressed.

If you want a one-word answer for what machines can never do, over the next decade at least, the answer is “abstract”. That’s a very human skill. Machines can be programmed to do it, but abstracting through following codes is not really abstracting, just as map making is not about following a list of instructions about what should and shouldn’t be on the map. Computers can only understand something they have been specifically programmed to understand. And nobody has yet managed to program a computer to abstract anything, beyond following specific programming.

There is a hit and miss element to human abstraction, if we make a guess of what a good abstraction would be and then test it on people. If it was purely hit and miss, a computer could do this too. But human abstraction probably is far more judgement than hit-and-miss.

Conclusion - authoritarian societies

So far, this book has focussed on making organisations better, and better able to achieve societal goals.

This book will conclude by outlining a further benefit to this approach of supporting domain experts to become citizen developers, and that is that organisations in non-authoritarian societies can be stronger than those in authoritarian ones.

Because being in an authoritarian society makes it very hard to do abstraction.

Abstraction is a form of creative thinking, which people can only do when they feel free to let their minds wander. It draws on the full powers of our brain, it is hard work.

When we are stressed or unhappy, part of our brain power goes into thinking about this. If we are disillusioned or unmotivated, we do not see the reason to put all the power of our brains into making our organisations better.

Script based thinking is like the default of human operations, what we do when we are tired and unhappy. It is also possible to force people to follow scripts through systems of punishments if they are not followed. It is not possible to force people to be creative, or look for new ways to achieve goals.

For authoritarian societies to be economically successful, they need organisations which can be managed and staffed by script followers.

This can include government organisations which have a brief to implement what the leader decides, not think about better ways to do something, and expect the public to follow the orders, under threat of punishment.

This can include businesses which do the same thing over and over again on a big scale, including in manufacturing, retail and services, to standardise business offerings as far as possible, and to do software development for big markets.

This discussion continues to the world of AI, which is economically successful when it can roll out the same tool thousands of times, and there are less concerns when it makes a mistake.

An authoritarian society can roll out a face recognition AI system in law enforcement easily, if it need not care about the people who are mis-recognised.

Much of what non-authoritarian societies offer, and authoritarian ones don't, could be seen as a weakness from a purely economic perspective. There is no clear benefit to a company's economy from providing state resources to care for elderly or disabled, for example. Countries do this because people see it as the right thing to do. Not all countries do.

If we are in a situation of a battle between authoritarian and non-authoritarian societies over who has the strongest economy, it is actually conceivable that the authoritarian one could win. And then it can access the goal based and abstraction thinkers by being the only pathway available for them to get an income.

The way for non-authoritarian societies to be stronger economically is to have stronger businesses, particularly in fields which do not do the same thing over and over again on a big scale.

We can see in 2021 that non-authoritarian businesses have a clear edge on authoritarian ones in areas of manufacturing which are intrinsically specialist and cannot be done the same way on a big scale, such as developing specialist machinery for factories. Also producing microchips, some areas of software design, education and defence.

Non authoritarian societies are better at making highly complex combustion engines, while the authoritarian ones may see that they can win in mass produced, simpler, electric cars.

Non-authoritarian societies also have an edge in areas which rely on complex risk management, including in financial, engineering and heavy industry domains. Although this only matters if you care about safety.