BSS transformation

Approach for predictable BSS transformation and business simplification

October 2023

RA Advisory

This document presents a structured, proven approach for full or partial core system replacement through greenfield transformation.

The described approach is focused on leveraging vendor capabilities to achieve competent execution and risk management, as well as simplifying business through the use of standard systems. In this way, the typically much-needed product and process simplifications become part of the core system replacement process. So, while the mechanics of the approach are focused on implementing systems, they include a business transformation in part as a prerequisite and in part as a product of the process.

The approach focuses on cost predictability, both in the project execution and in the subsequent operation and emphasizes the need for change management.

---

1 Introduction

Increasing cost and complexity of any non-trivial IT landscape appears to be a natural force that universally drives all companies towards a situation where "legacy" systems are both a cost driver and a constant barrier to business agility. Or, if not quite a natural force, then inherent in the way IT systems are often managed and evolving over time.

Replacement of such "legacy" systems in a predictable way, leveraging standard systems, is the topic of this document. Such replacement is here termed "transformation project".

It is a key assumption that the desired outcome is to (i) reduce inhouse system development to areas where it has potential to provide differentiation and value creation, and (ii) to leverage standard systems where possible, preferably with large functional scope. On the topic of the rational of standard systems, see further in section 6.8.

The document is primarily based on hands-on experience from core system (BSS/OSS) replacement projects in the telecommunications industry and the examples are taken from there. However, the discussion of approaches is also applicable in other industries that are highly IT dependent, and where a sufficiently mature vendor landscape exists. The approach for greenfield replacement requires firstly (i) that there exists a number of vendors that can support the core business with a standard system, leading to an advantageous competitive situation from the perspective of the buyer. And secondly (ii) that simplification of products, processes, organisation and other elements of everyday business logic is possible, permitting corresponding IT simplification.[1]

A central challenge in transformation projects lies in the complexity of legacy products and processes. Simply doing a "lift and shift" of all existing complexity will normally cause the new system to inherit the complexity of the old systems, thus reducing the potential upside significantly.

For this reason, simplification is an inescapable prerequisite for gaining full effect of system renewal. However, the system renewal can also be used as a lever to execute on this simplification. The approach set out in this document takes this focus on the relationship between simplification and system replacement.

The document is structured as follows.

Section 2 discusses the need for senior management commitment. This "top-level change management" is the foundation for projects of the character discussed here.

Section 3 outlines the alternative approaches for BSS replacement. The rest of the document is focused materially on the greenfield approach.

Section 4 discusses BSS architecture options and their relation to the implementation process.

Section 5 discusses target operating model and its implication on the implementation project.

Section 6 describes the first, preparatory phase of the greenfield approach where the requirements and contracts are delivered.

Section 7 describes the second phase of the greenfield approach, the implementation.

[1] This can be challenging in businesses with very long-running contracts. For such scenarios, a variation of the approach for simplification must be found.

Section 8 is a wrap-up of the greenfield approach.

2 Managing change

There are two main elements of change management pertinent to projects such as those contemplated here. One is the normal, operational change management of any project implementing new IT systems: preparing people for new processes, tools, and roles through training, including the related understanding of shifts in responsibilities and interfaces. This is discussed further in section 7.3.

While the normal change management is crucial, the commitment from senior management is a prerequisite for successful project execution. This section discusses this required management commitment to the complex changes major system replacements entail. This discussion starts with outlining some of the key problems that the execution of a transformation project will often face, leading to the concluding views on the role of the senior management.

2.1 Priority

In most organisations, the staff is fully occupied with the daily operation and the on-going incremental improvements that serve current and new customers and delivery of the immediate financial targets. Even if transformation projects are driven by external staffing, they invariably require significant internal participation. Both in the project themselves, but also in managing and delivering dependencies.

Priority is required for the sum of activities of transformation projects, but it is also required that the key staff involved have the seniority, competence, and tenure that enables them to define the future company. This type of core staff is invariably important to running daily business and therefore typically have management roles that they may be reluctant to exchange for a project role.

The approach outlined here, as well as other similar, attempts to provide predictability of the necessary effort, dependencies, and risks. Recognizing that such predictability is very difficult to achieve with the current state of the industry, most experience practitioners recommend a significant buffer or contingency in time and cost.

Even when formally having set aside such contingencies, activating it often leads to priority challenges, e.g., staff will not be released, other projects will need to yield to implement dependencies that were not originally identified. This requires the recognition that such challenges are inevitable and the willingness to deal with them.

2.2 Stability of requirements

The headline here sounds really waterfall-ish and in a sense it is. However, even when employing more modern methods, all greenfield transformation projects have a large-sized initial delivery. In spite of talking about "minimum viable product", a core system needs a lot of functionality before it can be released to serve its purpose internally as well as towards customers. In a brownfield approach, this can in some scenarios be done in smaller chunks, but it will be a balance between the "chunk size" and the number of interim interfaces one wishes to build.

One beacon that is often held out to how "agile" we ought to be is Spotify. There is a very entertaining cartoon-ish version of this (at the time of writing it can be found by searching for "spotify engineering culture"). As part of the story, they talk about what led to the culture and that includes that they went from a monolithic structure to the current model (where they basically structured the organisation and the core system in alignment). The story does not go into details on this move, but as they were already serving customers with a specific product, that must have set a certain minimum bar for the product they launched in the new structure. Hence the need for a certain complexity of the initial release. The point being that getting into the "model agile way of working" typically requires a large-sized initial delivery that is less well suited for a pure agile development method.

As this document is focused on implementing standard systems, not developing software, the initial release in a transformation project will have a significant size, and while it can be tested in smaller sizes, it does not meet reality until release time. It can also be released gradually to customers; however, such gradual release is normally not tolerable to last for too long time and has its own complexity.

In summary, implementing administrative standard systems in existing businesses like telecommunication requires a large initial delivery that takes a significant amount of time, normally at least two years from start of procurement to initial release.

The world is not frozen for this time, and a transformation project needs to adapt to necessary changes. However, if those changes become excessive it threatens not only the timeline but the entire success of the transformation project.

One key mechanism contributing to this phenomenon is the following. In well-conducted transformation projects, decisions are recorded to form the basis for subsequent phases. The sum of decisions over time becomes a collective understanding in the transformation project, which is a sort of benign inertia that keeps the project on its course.

What is not normally documented is the set of discussions and rationales that led to each decision. In practice, most non-trivial issues are resolved in informal cooperation, e.g., through collaboration tools in combination with problem-solving meetings (of which minutes are rarely taken beyond what was actually decided). Documenting the trail of logic in a manner that makes it possible later to retrieve it effectively and reliably is in practice impossible.

The consequence is that understanding the implication of a proposed change requires revisiting decisions and attempting to collectively recollect the rationale that led to the decision and redo that in context of the proposed change. This upsets the "benign inertia" of the transformation project, and in case of many such upsetting events, the transformation project loses its collective sense of direction.

Managing this is further complicated by the fact that there is no KPI or other hard facts indicating how close to the tolerance limit a transformation project is getting, making it hard to withstand a change that is deemed necessary to accommodate for short-term business reasons.

Recognizing again that changes are necessary to accommodate, any suggested changes should be filtered thoroughly in a formal process to ensure that a balance between the requirements of the on-going business and the progress of the transformation project may be found.

2.3 Overlap with line organisation

The type of transformation projects discussed here, if they are to have any chance of being successful, will "take over" substantial proportions of management as illustrated in Figure 1: *project overlap with line organisation* below with generic organisational units (IT, products, markets, finance):

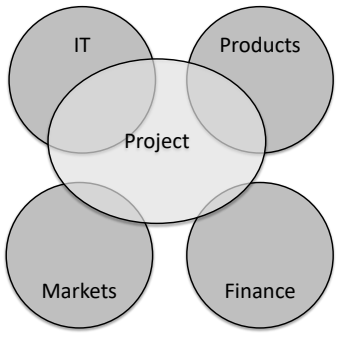

Figure 1: project overlap with line organisation

The project will define a significant proportion of the future business: IT, processes, products with impact on revenue, organisation. This means that the scope of the line management will be reduced for the duration of the project, potentially significantly.

It is important to recognize that this is the nature of the transformation project, not just an avoidable turf war, and that line management decision mandate must be exercised through different means than the direct command for areas that are "taken over".

2.4 Management commitment

In summary, in order to be successful, transformation projects will take staff from line organisation, in part take over decision mandates and in addition limit the amount of change that line organisation can implement for the duration of the project.

Furthermore, the transformation project tends to be less than fully predictable, causing the limitations on the line organisation to extend in ways that were not originally envisioned.

All in all, a transformation project is a major pain and disruption to the way normal business is conducted, making the ability to meet the short-term financial targets (for which there is not normally very much amnesty) even harder. All of these factors will breed resistance, often growing over time.

Executing a transformation project, therefore, requires that the senior management team uniformly and unequivocally supports it. This support will only come if top management collectively recognizes the imperative for the change.

The organisation needs to understand that the execution of the project is mandatory, and the senior management team must be motivated by implementing the change and the corresponding transformation. Furthermore, the management team (as well as the board and other ownership stakeholders) must be committed to execution, including taking out products and corresponding revenue that drives excessive complexity or precludes using standard systems.

For these reasons, it is also typically best if a transformation project refers directly to the CEO.[2] In case the transformation project reports to a CIO/CTIO or transformation officer, such person

[2] Note that "CEO" here is intended to refer to the head of the business unit undertaking the transformation.

and role must be respected, business focused and backed up fully by the CEO.

The CEO reference corresponds to the classical PRINCE2 model with the CEO as chairman and the respective management team members filling the relevant roles. In the CIO reference, the roles are less obvious, including who chairs the steering group.

3 Categorization of approaches

This section outlines the different approaches to IT transformation as a preamble to focusing on the greenfield approach.

The IT landscape of a typical company consists of many components, some which are tightly coupled and others which are loosely coupled. The level to which systems are coupled is individual for each company. For example, in telecom companies, ERP and HR systems are typically loosely coupled with the BSS system, whereas CRM and order management within the BSS system typically are tightly coupled.

The discussion of approach here relates to replacement of a group of tightly coupled systems. For example, if the CRM, order management and billing systems are implemented in one monolithic system, they are very tightly coupled, and the discussions on pros and cons below apply. Conversely, if the ERP system is loosely coupled from the BSS, replacement of these two components can happen independently without too much friction and the discussion below therefore does not apply.

While a number of approaches are possible for replacing a group of tightly coupled systems, most are variations of the following:

1. Brownfield, where parts of the tightly coupled systems are replaced with a new system.
2. Carve-out, where parts of tightly coupled systems are moved gradually to new components.
3. Greenfield, where tightly coupled systems are replaced with a new system built initially alongside the legacy system stack.

Each approach is described briefly in the following subsections.

The greenfield approach, which is the primary focus of this document, is described in a lot more detail in section 6-7.

3.1 Brownfield approach

In the brownfield approach, parts of the legacy landscape are replaced with a new system. Typically, this is a replacement of a CRM, online or billing system, but can also be smaller components like mediation, output management.

The approach is to essentially replace an existing component with a new one, supporting the same set of products and processes as previously.

For small components, this is fine, as these generally do not carry the full complexity of the product portfolio and has comparatively simple and often standardized interfaces.

Brownfield may also for larger components be a viable approach in cases where the inherited complexity can be managed from both a product and an interface perspective. Since this would not normally be the case for non-trivial IT stacks, a thorough review should be undertaken prior to embarking on brownfield replacement.

The interface perspective reflects the number and complexity of interfaces affected; often hundreds of interfaces need to be built or modified. Simplicity of interfaces would normally require that the system stack is structured in layers in a modularized manner with well-defined and centralized API separating the various layers, essentially ensuring loose coupling.

The product perspective includes limiting the product support of the new system to only the newest generation(s) of products, leaving legacy products to be serviced “somewhere else”. For instance, when replacing CRM, the billing system may have rudimentary operational CRM capabilities that may be used for legacy products while the new CRM is used only for newer products.

The viability of the brownfield approach is highly dependent upon the specific context of the company as well as the components subject to replacement. The specific complexity should be well-understood prior to initiating the brown-field approach.

3.2 Carve-out approach

In the carve-out approach, functionality in the legacy systems is replaced by moving it gradually to new components. The approach resembles the brownfield in the sense that replacement is partial, and a lot of interfaces typically need to be built. In the best applications of the approach, the new components only support a subset of offerings, ensuring a clean-up in the process. Transferring all legacy complexity gives the same problems as the corresponding approach for brownfield replacement.

Replacement of components can secure reduced impact through back-propagating transactional updates to legacy components, causing new and legacy components to contain the same data and

thus reducing impact on surrounding systems (and thereby reducing the number of changes for each carve-out iteration). Since upholding legacy components defies the purpose of replacement, this is a migration approach, not viable for longer term. For spreading the impact of changes, both in risk and cost, out in time, it can be an effective migration strategy.

The approach can, dependent upon the specific application landscape, have the advantage of providing partial results during the process. This is an important risk-reducing factor as the stamina required for complex system replacement can be challenging to uphold. As a gradual approach, it can also reduce the IT risk since components go into production gradually.

The key challenge with the approach is to find a good sequence to conduct the carve-out. If the carve-out becomes too complex for each component, it will have the challenges of brownfield. If the number of components required to replace to make integration manageable is very large, it will effective be akin to greenfield.

Since the components to be replaced often will not adhere to the boundaries of standard systems, the approach can require quite substantial amount of custom development, not all of which is temporary. This can significantly reduce the value of the replacement, since complexity is more likely to increase gradually in a custom system.

As for the brownfield approach, the carve-out approach is highly dependent upon the specific context.

3.3 Greenfield approach

The last option considered here is the greenfield approach. In the greenfield approach, a new stack is built alongside the old stack with suitable integration points allowing for dual operation. Following the setup of the new stack, customers are migrated into it.

The greenfield approach has the advantage of enabling simplification and - if properly executed - tends to be more predictable.

The greenfield approach is not discussed further here as it is the topic of the remaining document.

4 BSS architecture

As noted above, a core belief underlying this document is the use of standard systems. This has several implications.

Firstly, as far as the standard systems are concerned, software development and the internal software architecture is of limited importance, provided that the vendors can be expected to follow overall industry trends.

Secondly, the key architectural focus is on avoiding complex integrations as these tend to be error prone and costly to both implement and maintain.

Thirdly, since custom development can be limited to comparatively few and limited areas, the complexity of these can be limited.

Following a slight detour on monolithic architecture, this section outlines some architectural patterns that can help avoiding excessively complex integrations.

4.1 In defence of the monolith

Some of the architecture illustrations below have large functional clusters from a single vendor, resembling a "monolithic" system architecture.

In most telecom architecture discussions, the term "monolith" is used derogatively, and almost synonymously, for an application with poor modularization where all changes are highly complex.

Understood as such, obviously the monolith is not a good thing. However, understood as a pre-integrated system covering most functional requirements, based on individual subproducts with distinct functionality, it enables the business while avoiding the need to become a software company. Implementing standard systems with small footprint typically requires quite extensive integration since (i) it tends to become more like a brownfield approach and (ii) the small footprint generally implies that multiple standard systems are implemented at the same time. Thus, implementing standard systems without extensive integration effort in practice requires quite extensive footprint.[3]

Procuring an extensive core system from a single vendor makes the on-going operations a lot simpler. The number of releases to be co-ordinated are smaller, there is limited retesting and rebuilding of interfaces, and the responsibilities are clearly defined.

[3] This does not apply in case a number of systems are implemented that are pre-integrated with a commitment to remain so. Such a situation will lead to a more complex sourcing situation, but the implementation is akin to having one system with a large footprint.

Obviously, such extensive footprint results in extensive vendor lock-in. However, the vendor lock-in also exists where multiple vendors have smaller footprint, and the vendor lock-in can be managed. See section 6.7 for further discussion on the vendor lock-in topic.

4.2 Patterns

This section outlines architecture patterns that work well for systems with classical delineation for telecom companies to provide BSS and OSS system support without becoming a software company (internally or outsourced).

The purpose is to illustrate what the scope could be when the sourcing process focuses on maximizing vendor footprint. This is done by showing patterns that provide for simple integration as well as patterns that requires complex integration.

The purpose is emphatically not to provide an exhaustive list of BSS/OSS patterns nor to be normative on what architecture patterns overall are workable.

4.2.1 Maximized scope

In this pattern, almost all BSS/OSS functionality can be found with the main vendor. It can be illustrated as follows:

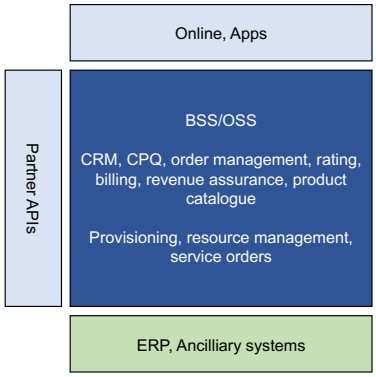

Figure 2: Maximized functional scope

The dark blue is a full scope supplied by a single vendor, the light blue customized and the green outside of the BSS/OSS scope.

Obviously, it is possible that both partner APIs, online and apps are supplied by the main BSS vendor. Normally, the online parts of the BSS systems are not satisfactory from the customer friendliness perspective, focusing more on the transactions than the experience. But there may be exceptions where this is not the case.

A 360 degree customer view is ensured by absence of data storage in the online part.

Assuming that the vendor implementation is standard, the pattern permits a very simple operations in that all material development is outsourced to the vendor of standard software.

Note that the pattern assumes that the core BSS can handle all requirements for both B2C and B2B.

4.2.2 Separate OSS

This scenario is the same as the first pattern set out in section 4.2.1, except that the OSS (similar to production in the ODA parlance) is separated.

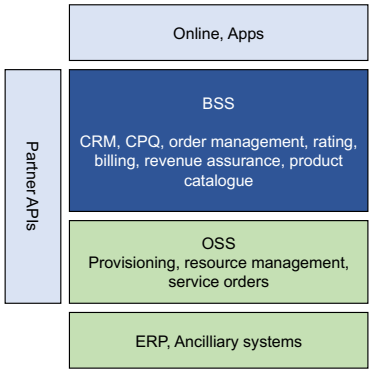

Figure 3: Maximized BSS scope

The dark blue is a full scope supplied by a single vendor, the light blue customized APIs and the green outside of the BSS scope.

In this structure, the ODA production and classical OSS are similar in scope (the OSS delineation is more open than the ODA production).

4.2.3 B2B sales tooling separate

This scenario builds on the "separate OSS" pattern in section 4.2.2, but whether the OSS is separate as in section 4.2.2 or from the same vendor as the BSS as in section 4.2.1 is not material for the discussions in this section.

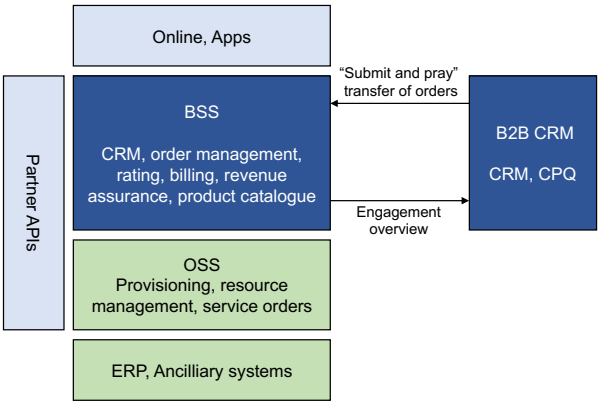

Figure 4: Separate B2B CRM

Here, the B2B sales process is supported by a different tool. This pattern is not unusual since the B2B functionality in classical BSS stacks often is deemed inadequate.

The pattern can work under a few critical assumptions:

1. The CPQ in the B2B CRM is held within what can be handled in the billing and product catalogue of the main billing system.
2. The orders formed in the B2B CRM are submitted to the main BSS once and then executed and maintained in the BSS.
3. An engagement overview can be taken from the BSS into the B2B CRM.
4. No further integration takes place between B2B CRM and BSS.

With these assumptions, the B2B sales processes can be handled in the B2B CRM and the customer management in the main BSS. The specific complexity depends on the specific systems, but in most cases, these are quite simple.

4.2.4 B2B CRM as front-end

This is a variation of the above where the interface between the B2B CRM and the BSS is a full integration. This violates the critical assumptions listed in section 4.2.3.

This pattern is to be avoided for the following reasons:

1. The integration between the CRM and the core system is normally highly complex since the number of states in, e.g., orders, products, payment states, are very high.
2. Both systems (BSS and B2B CRM) typically view themselves as "data masters", adding to the complexity of the integration.
3. Such complex integration normally leads to data inconsistencies that can impact customer experience.

The only exception is in case the vendors of the B2B CRM and the BSS have pre-integrated their systems. This ensures – or at least outsources the risk of – data inconsistencies and ensures that the integration is upheld in the face of new releases.

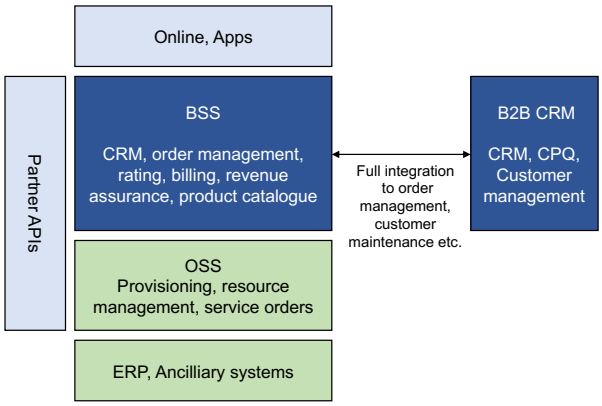

Figure 5: B2B CRM as BSS front-end

4.2.5 Separate B2B and B2C stacks

This pattern is identical to the one set out in section 4.2.2 except that B2C and B2B are managed separately.

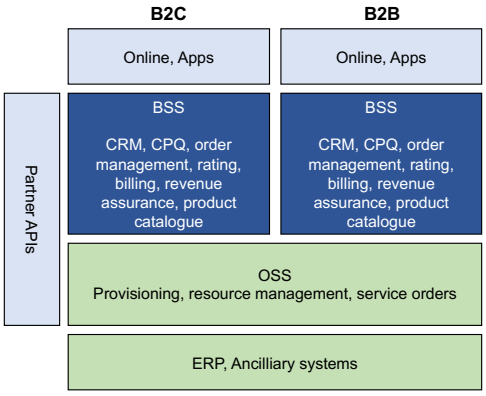

Figure 6: Separate B2B and B2C stacks

From an architecture perspective, this works fine. It can cause challenges for retail (where applicable) since they would need to use two different systems depending on the type of customer.

Also, some complexity may arise from the fact that small companies as customers tend to move between B2B and B2C offerings.

Finally, unless special circumstances prevail, it is likely to be a more expensive solution as most BSS vendors support both B2B and B2C.

In this scenario, the two stacks will be subject to separate sourcing processes.

4.2.6 Joint billing

This pattern is similar to the "separate B2B and B2C stacks" of section 4.2.5, except that it has a joint billing system across B2B and B2C.

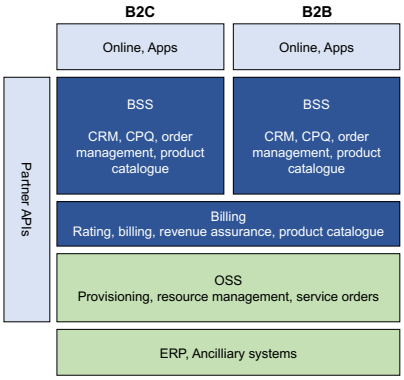

Figure 7: Joint billing

The integrations here are less complex than the ones set out in section 4.2.4 but still require a translation between product catalogues that can either be complex integrations or somewhat tedious manual updates across systems. As above, if the systems are pre-integrated by the vendors, this becomes a smaller, potentially even marginal, issue.

The pattern is not very obvious as most BSS vendors include billing in their package and the integration still has some complexity.

Also, the sourcing process increases in complexity as B2C and B2B both need to integrate with the billing, implying three processes in parallel for a full BSS replacement. Running them sequentially (in case this was also the architecture pattern prior to the transformation project) is possible but will entail at least some of the challenges discussed in section 3.1 on the brownfield approach.

4.2.7 Summary

As noted initially, this section outlines a few patterns for illustration of what to aim for and what to avoid. For all scenarios, avoiding data in the channels provides for consistent cross-channel view of customers.

Scenarios with heavy integration do not lend themselves to scenarios where the target is to avoid in-house development.

The patterns evolve from a "best of large suite" to more gradual "best of suite" with the joint billing model set out in section 4.2.6 moving towards the "best of breed" scenario. The general view taken here is that the complexity added from "best of breed" is rarely justified by differentiating capabilities.

The process defined in section 6 and 7 sets out to select and implement such a pre-integrated system as the backbone of the system landscape. This is back to the basic assumptions of the document, i.e., that the target is to avoid in-house development except where it really has a differentiation potential.

5 Operating model

Once the implementation project is completed, the system needs to be managed to support the business. This includes the day-to-day operations as well as the on-going enhancements.

To a large extent, the project roles set the direction for the operating roles, e.g., through competence building. In addition, the structure of sourcing the solution is influenced by how the target operating model is. Therefore, understanding the target roles is important as part of setting up the project and executing the initial sourcing process.

This section sets out a few considerations that are necessary to include as part of the implementation project in preparation of normal operation following the transformation project. It does not aspire to describe a full list of items to consider for IT operations.

5.1 General considerations

The following taxonomy for the technical operating model is used.

	Function	Description
AD	Application development	The activities required for larger implementation efforts, e.g., new VAS services.
AM	Application maintenance	Activities required for bug fixing or minor, incremental development efforts.
AO	Application Operations	Day to day execution of applications, e.g., billing runs, data fixes, monitoring.
BO	Basic Operations	Implementing and supporting hardware, basic software, database software and the like.

Figure 8: Operating model taxonomy

For standard software, all AM and AD for the application itself is delivered by the software vendor through the support and maintenance agreement. These activities are natural monopolies in the sense that as owner of the software, only the software vendor is capable of providing these services. The

support and maintenance service, fortunately, can be contained in cost through the contract.[4]

On the other hand, the operations of the application as well as the ongoing configuration and enhancement outside the core software are not natural monopolies. These areas cannot readily be contained commercially as their scope is less clear. While it is possible to outsource these services directly following the implementation project, the result will be that limited hands-on competence exists internally and, in consequence, changing the sourcing setup can be challenging. Upholding commercial leverage in such a situation is not realistic and may even spill over to areas that initially are contained well.

Further considerations on the "best" model can be found in section 6.4.

5.2 Cloud

As more applications are "moving into the cloud", it becomes an important discussion when setting up the operating model. This is discussed further below.

5.2.1 SaaS and IaaS

In discussion "cloud", it is important to distinguish between the different models, in particular the "software as a service" (SaaS) vs. the "infrastructure / platform as a service" (IaaS).

For SaaS, the operation is inseparable from the application and other operating models are not possible. For such systems, the typical license model can be made predictable as it often is linked to user counts, revenue under management or similar. This can, therefore, be managed through the sourcing process.[5]

IaaS is materially an outsourced basic operation (BO) with substantial flexibility in available capacity. For IaaS it is typically challenging to get predictability. This is not quite unreasonable, as the load on the platform depends on the use by the customer. Since many software vendors themselves outsource IaaS to the major, global suppliers (Microsoft, Google, Amazon), the customer typically "inherits" these terms. The commercial leverage in this situation is typically non-existent in a direct negotiation and must therefore be secured in a different way, i.e., through the ability to switch sourcing model.

5.2.2 The imperative of the Cloud?

When reading the literature of vendors, TM Forum and others, it often appears as if moving to the cloud itself is a prerequisite for tapping into new revenue streams (although the actual revenue streams typically are quite vaguely described). Even better if one is "cloud-native", which apparently is a combination of microservices, access to flexible capacity and CI/CD.

When reading statements, e.g., top-50 in a google search, it appears that the industry has accepted these as almost axiomatic truths that do not need justification.

The view view taken here is that when considering the options, one should look to identify the real advantages and rather than abstractions, look for real-world customer problems that may be solved and for which solutions customers are actually willing to pay.

A few specific comments on cloud pertaining to the operating model are listed below.

SaaS

A SaaS application has the advantage of being simple to onboard and its universal upgrade model promotes some discipline in how it is used. Additionally, it can simplify operations as it represents outsourcing of a significant operational footprint (not unlike the "monolith" of large functional scope). Such simplification requires that the overall architecture remains simple without too many integrations.

Some SaaS solutions put tighter limits on what customizations can be done, thus promoting discipline. However, this characteristic is not universal and not really linked to the SaaS model (some systems with more classical licensing model has the same characteristic). For others, the customization can still be quite massive.

For smaller businesses the SaaS model has the advantage of a quick start without up-front cost and flexible scaling. These advantages are more dubious for stable businesses and for such it is the

[4] Having the software vendor do support and maintenance at a fixed price is the general situation and target; however, excessive customization may complicate this.

[5] With some of the larger SaaS vendors, it is very difficult to achieve predictability in pricing as they reserve the right to adjust pricing unilaterally. This is addressed in principle in the process, but obviously the process cannot in itself make a vendor adhere to certain terms. However, it is important to maintain that there is nothing in the technology that precludes commercial predictability, even if SaaS vendors try to convince customers of this. As an aside, anecdotal evidence indicates that being careful in understanding the charge model for the large SaaS companies is important as quite a few companies have been surprised by the resulting cost levels from seemingly innocent charge models.

continued upgrades (which can be an advantage but may also turn out to be a disadvantage in case functionality disappears) and cost. The latter can be ascertained in the procurement process, even if some of the SaaS charging models can be very challenging to understand.

Typically, using SaaS also moves costs from CAPEX to OPEX.

IaaS

Moving to the cloud in the sense that it is a hired infrastructure provides similar advantages as SaaS: it scales well and starting is easy.

Since it is a rented infrastructure, it also shifts costs from CAPEX to OPEX, typically a disadvantage for telecommunication operators.

Again, this may make sense depending upon the specific situation, but there is also a movement away from this trend. Just try searching for "why we are leaving the cloud" – at least some will say that they no longer need the scalability, the cloud comes with high cost and the cloud has its own complexity.

Cloud native

There are various definitions, but most include microservices as their central theme. And state various benefits like increased efficiency, reduced costs, and availability, again typically without qualification.

As this is not a discussion on software architecture in general, the pertinent question is whether it is important that the BSS vendor uses microservices, Kubernetes and is cloud native. Unless you have specific requirements that can translate into BSS requirements it comes down to cost, including the cost of the operational complexity.

6 Greenfield phase 1: Contracting

This section defines a methodology for the first phase of the greenfield approach, which seeks to shape the project and handles the process up to the point where a contract is signed. The contract is the main physical manifestation of the phase, but following the process ensures that the contract enables a simplified, standard-based business support.

The desired characteristics of the approach include:

1. The process should be designed to ensure business and system simplification.
2. The process should be designed to maximize the use of standard systems.
3. The process should secure short- and long-term cost efficiency and commercial leverage.
4. The process should not require a detailed specification in order to start the procurement process.

This contracting approach section contains the following subsections that are part of or background to the approach.

1. The importance of simplification and how the contracting phase goes hand in hand with the business simplification.
2. Discussion on why an RFP is considered a necessary first step in a transformation project.
3. Discussion of the 'specification trap' and its implications are outlined. Avoiding this is defining for the suggested approach, including in particular the "principle of partial specification".
4. The target operating model and its role in securing continued commercial leverage.
5. Specifics on the maintenance agreement.
6. Considerations for cross-border synergies.
7. Vendor lock-in.
8. Definition and importance of standard systems.
9. Definition of terminology for project roles.
10. Discussion on whether system integrators should be invited early in the process.
11. Identification of potential vendors.

Following these introductory topics, the main methodology for the contracting phase is outlined.

The in-project concept of "people-related change management" is described in the second part of the approach, in section 7.3; the framing of the project including securing the organisational stamina for execution, is discussed in section 2.

6.1 Simplification

Simplicity cannot be emphasized sufficiently; empirical evidence shows that length of operation and simplicity of product are the only significant explanatory factors impacting IT costs for mobile operators. While not directly documented in benchmarks, it appears obvious that complexity in IT and complexity in business is a vicious circle. Complex business requirements drive complex IT systems; which drive long development times; which cause short term solutions that tend to live very long; which causes process faults and manual workarounds; which impacts customers and calls for 'short-term' solutions. Along the way, the IT staff keeps talking about "technical debt", which is not addressed until it becomes so massive that a transformation project is initiated.

The agenda of simplicity should therefore focus on making things simple and keeping them simple (the latter is not part of the discussion here).

6.1.1 Start from scratch

To make sure that all current customers are served with relevant products, one need to look at the current products. However, if one starts with current products and processes, one risks getting stuck in current functionality and complexity (obviously depending upon the starting point).

So, to achieve simplicity, the input should be the main segments, products in the market serving those and from there derive the products to offer in the future. In other words, a process like one would do in a new company.

Once this has been developed over a few iterations, the current product portfolio can be used to check if there are important features or whole products that have been left out.

6.1.2 It is a two-way street

Laying the foundation for business transformation and simplicity is a separate piece of work as one needs to design a new business.

However, a key to simplicity is to use standard systems. Such systems come with version 20+ of data models and process support with input from many implementations. There is no way a similar quality can be achieved in a design made from scratch. The required experience cannot realistically be brought to bear in a specific project. For a standard system, the value deteriorates quickly if extensive customizations are made: the ability to follow the upgrade path, utilize new features and avoid own development will be lost.

It is therefore essential to adapt to the capabilities to standard systems, unless the value creation of not doing so is clear and the consequences in terms of cost (long term as well as short term) and operating model is understood and accepted.

Furthermore, the requirements specified in the initial phase of business transformation should recognize this fact and try to keep a fairly high level of abstraction as well as focusing on differentiating requirements; for instance, the ability to rate an SMS or produce a basic bill is not likely to be missing in any system today, whereas hierarchical split billing handling the relevant tax quirks may not be standard.

One reason that the approach outlined here advocates starting from scratch is to facilitate this two-way thinking: define the products with which to serve the customers and use that, rather than existing functionality, as the starting point. And in doing so, bear in mind that product is a lot more than raw telecom functionality and price. Product is also how the customer experience the processes (and, implicitly, data), e.g., through flexibility and usability of self service.

In the approach outlined in sections 6-7 of this document, the two-way street view is very explicit.

6.2 Why the RFP?

The classical RFP purpose is a pure procurement thinking: Getting the best product at the best price. That still makes sense, as the RFP provides a structured way of asking the market on fulfilment and pricing.

In addition, the RFP as outlined here, provides a way of adjusting the requirements to the specific application, to the effect that it supports the business in the best possible way. It furthermore tees off the subsequent phase in designing the process and ensuring aligned expectations.

Finally, the RFP ensures a fixed price. Pricing is a prerequisite for comparing alternatives and executed properly, fixed price gives predictability. Further, time and material contracts are described as something to avoid. This is the view taken here for all vendors involved, with the system integrator (see below for the various roles) as a potential exception. The reasons are briefly outlined here.

Fixed-price contracts do not necessarily give cost savings relative to time and material contracts. But they can enable cost predictability.

More importantly, time and material projects lack the tension of keeping scope, keeping to standard and focusing on finishing in time, factors which in fixed-price contracts are driven strongly by vendors.

Also, the fixed price contract ensures that vendors activate their internal risk management apparatus. This is an important assurance when embarking on such a project. In time and material, the risk is normally entirely with the customer, and the vendors will be less concerned with risk management – if the project runs longer, the vendor will gain more revenue.

In other words, fixed price promotes discipline in that the vendor will issue formal requests in case of new requirements. These will, therefore, be scrutinized carefully. Also, the vendor will not suggest extensions or “nice to haves”, as these are unlikely to be approved and therefore will need to be delivered for free. This “benign pressure” that forces discipline on both sides is not possible without a fixed price and a strict change process.

6.3 The specification trap

This section outlines the concept of the "specification trap", which is the dilemma of how to contract a delivery that is not fully specified yet getting a predictable pricing. This dilemma and its (partial) resolution is defining for the method outlined in section 6.12.

6.3.1 Partnering

Before diving into the "specification trap", a note on the relationship between a vendor and a customer: not infrequently, the notion of "partnering with the vendor" comes up as an alternative to running a sourcing process, in full or in part. Vendors often promote this notion.

The perspective taken here is that the interests of a vendor and a customer are fundamentally opposed. The relationship can be fine, constructive, even friendly and beneficial for both. But coming down to essentials, the customer has an interest in getting as many services for as small a cost as possible. And the vendor has the opposite interest.

This is not to say that partnerships do not exist. One real-life example experience was an Asian mobile operator who agreed with a RAN vendor to pay a percentage of revenue for implementation and operation of the RAN. In this way, the vendor's income is linked to the results of the customer and the opposing interests are less pronounced. But such deals are not commonplace and almost unheard of in the IT system space. [6]

The implication of this point is that one must secure oneself commercially or the vendor probably will take advantage of the situation. Not all vendors will do so at all times, but people and ownership changes, financial pressure fluctuates, and relying on partnerships can be fragile in such situations.

6.3.2 Waterfall

Now for the "specification trap". In a traditional waterfall approach to system procurement, the RFQ/RFP processes would attempt to specify all requirements in detail, sometimes in fairly extreme detail. This approach is not employed frequently anymore but serves as illustration of the 'specification trap'. The principle for specifying requirements in this way is illustrated below:

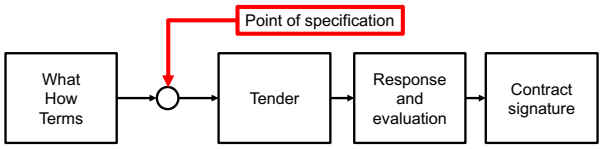

Figure 9: Traditional waterfall point of specification

The "point of specification" refers to the point in the process where the specification of the requirements are fixed. The process above is intuitively fine: the customer asks the vendors for a specific solution, vendors respond, and the best fit is chosen.

There are two important problems with this. The smaller problem is that it does not facilitate use of standard systems. The vendor signs up to specifications irrespectively of whether it is standard or not.

The larger problem is that it is not in practice possible. The amount of detail that needs to be written down is prohibitive. It is never right in the first place, and it is quickly outdated, sometimes even at the time of submission.

In addition, it is wasteful since one must specify requirements so standard that they are trivially fulfilled by any competent vendor.

6.3.3 Analyze - build

An alternative model is to engage with a vendor, typically a system integrator, and run the process as follows:

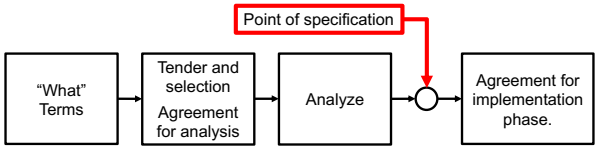

Figure 10: Point of specification after contracting

Here, the vendor is selected before the point of specification and assists in the analysis that leads to an agreement for the implementation. From a content perspective the model is fine, but it causes the customer to lose commercial leverage almost immediately. Essentially, it becomes a time and

[6] One example taken from the airline industry can serve to illustrate the difference. The example is from when "free" meals and drink were still a standard part of most flights. An airline catering company supplying to both traditional airlines, that included free drink and meals in their fares, and low-cost airlines where everything was payable. Their relationship with the traditional airlines was generally fine but burdened by the fact that discussions always focused on reducing cost (and thereby at some point the quality of the offerings). Whereas with the low-cost airlines, the focus was on how to sell as much catering as possible, as both the catering company and the airline benefited from such sales. The former relationship was fine but the latter had more characteristics of a partnership.

material agreement, at least as far as the system integrator goes.

6.3.4 Agile to the rescue?

Some vendors would argue that the answer to the dilemma lies in an 'agile' approach. While there are many advantages to an 'agile' approach, from a commercial perspective it still boils down to payment based on time and material. The 'point of specification' is therefore merely dragged out, but still after contracting. In a picture similar to the ones above, it looks as follows:

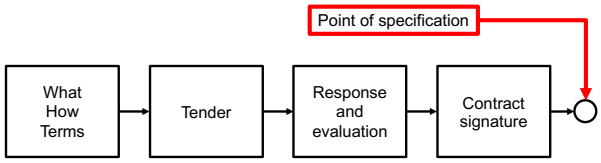

Figure 11: Point of specification in agile

Essentially, the specification is not really final until the project is done.

Agile certainly has a role in transformation projects. Most vendors will use agile methods in their implementation, and that makes a lot of sense. But contracts are in their essence not agile.

6.3.5 Addressing the specification trap

The process set out in this document, in particular section 6.12, attempts to address the 'specification trap'. The approach is illustrated below:

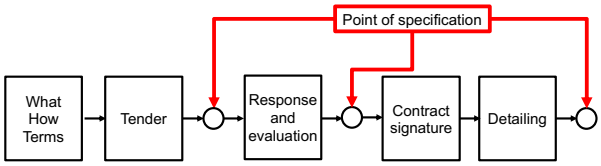

Figure 12: Multiple points of specification

The approach is to define the specification through the process in co-operation with the vendor before and after contract signature. The detailing that happens after contract signature needs to be subject to a set of rules set out in the contract. The key vehicle to this detailing is the principle of partial specification illustrated below.

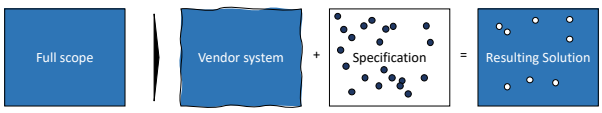

Figure 13: The principle of partial specification

This central principle is where the prerequisite of a competent vendor landscape for the business is used.

The principle is, reading from the left:

1. There is a full, but unknown scope. Unknown in the sense that it cannot be described, not that it is non-existent.
2. The vendor has a system that supports similar businesses with materially the same scope.
3. In order to run the process, a specification is written. The specification is, as the dots illustrate, only a small proportion of the full scope. In the illustration it is shown as random dots, but of course the text should focus on key processes and differentiating capabilities.
4. The resulting solution materially fulfils the full scope – exceptions as illustrated with the white dots must be handled either through workarounds or other systems. Or by simply just abandoning the requirement.

This can be contracted in the sense that the risk of completeness, at least in part, can be sourced to the vendor. While they may initially object, some of them forcefully, such agreement can be reached.

6.4 Target operating model

One aspiration for the approach outlined is to uphold commercial leverage with the customer. Clearly, it will be stronger before contract signature, but it is possible to keep it throughout the relationship.

The key to securing this is in the operating model following implementation. The primary software vendor will in practice have monopoly for the software. This is due to extreme shifting costs, as shifting to another software vendor entails yet another program like what is outlined here. Therefore, the future relationship with the software vendor should be characterized as follows.

1. The "monopoly" part, i.e., the continued maintenance of the core application, should be subject to fixed, predictable prices that are valid for perpetuity.
2. All other parts should be managed within the customer organisation, or subject to tender. This requires that the competences are not monopolized.

For these reasons, the product vendor should ideally, following acceptance of the primary delivery, be confined to delivering support and maintenance services. The other services required to operate and develop the solution should be in-house or sourced

to other vendors. This requires that competences are built with the customer throughout the project and that sufficient IPR is vested with the customer to permit maintenance of all relevant items. This way, the 'monopoly' component from the primary vendor is contained commercially and the other services can be subject to continued tendering.

The approach above is only possible if the solution remains a standard system as described in section 6.8 below.

The alternative, where the primary vendor continues to deliver development and maintenance services, extends the monopoly to areas where the cost cannot be pre-agreed. Such an approach makes the long-term cost predictability challenging.

6.5 Maintenance agreements

In section 6.4 above, it was stated that the maintenance agreement should have prices that are valid for perpetuity. This section expands on this and a few other key aspects of the support and maintenance agreements.

Most software vendors will attempt to argue that the termination notices should be "balanced", by which they imply equal timeline. However, equal timeline is nowhere near balanced. For the customer, the time from decision to replacement is a minimum of four years, in many cases longer. And the old system is likely to linger a bit longer for "edge cases". For the software vendor, it is a loss of revenue with no operational impact.

In practice, permitting a termination notice from the vendor of less than eight years after full go-live reduces long term cost predictability significantly. The eight years come are the sum of implementation and decision cycle. The implementation is typically at least four years. The decision cycle is the time it takes to start an implementation program – typically, the fact that a vendor raises prices will not shift internal priorities to system replacement quickly. Conversely, there is no reasonable business reason to have termination period from the customer longer than a year.

A second point on the support and maintenance agreement is to be careful with upgrades. Firstly, they should be included. Otherwise, the price control of the monopoly will, again, be eroded. Secondly, the assurance that all functionality is maintained in the upgrades, irrespective of the way the system is packaged, should be included.

A final point is to secure a "light-weight" version of the support and maintenance agreement following the decommissioning of the software system. Typically, the benefits of this will be with someone else than the people implementing the system, so it tends to get less attention. But retention of records can be a real pain without having a system to do it with, more so with GDPR requirements. On the other hand, one does not wish to pay full support for a system with one or two users, to which access is mostly theoretical.

6.6 Cross border synergies

Cross border synergies are discussed here as they, if desired, should be designed into the contract.

The discussion of cross-border synergies has an appealing logic ("why do we want to maintain X system stacks when we can make do with one"), which assumes a context of commonality that is typically not present. Furthermore, also similar to outsourcing, it is not supported by empirical evidence: scale in itself has limited value.

In most situations, the governance of products and processes is decentralized and the requirements to IT are therefore not co-ordinated. Not infrequently there are real market differences, especially for large telecom operator groups with operations in very different geographies. Centralizing the IT people servicing a number of different environments does not give significant scale advantages.

Only in situations similar to where outsourcing makes sense can cross-border synergies provide significant advantages: with high degree of similarity through use of standard systems or centralization of governance of requirements, are there significant advantages of having joint systems across borders. For this reason, the back-end systems like billing, mediation and rating are easier to obtain cross-border synergies for than customer facing systems like CRM and online.

In case the new system should support multiple business units, the governance of requirements, releases and relationship to other systems should be in place early, preferably before embarking on the contracting approach.

The question of cross-border synergies is the topic of another white paper available on the web page set out at the end of this document.

6.7 Vendor lock-in

The approach outlined here attempts to leverage vendor competences in a number of ways, including maximizing the scope of the product vendor. If successful, the product vendor will have a large footprint in the application architecture. This fact sometimes leads to concerns over vendor lock-in. This section discusses this concern and how it may be dealt with.

Firstly, vendor lock-in is unavoidable (unless one builds systems internally, in which case the "lock-in" is transferred to critical employees which is usually even worse). The lock-in does not become much worse from a large footprint. Any vendor with a substantial footprint is a challenge to replace, typically requiring 3-4 years from decision to execution. Therefore, vendor lock-in must be managed, as it cannot be avoided.

The approach suggested here is to secure that the areas where the product vendor has monopoly is limited to areas where the cost can be predictable. Specifically, securing that all work following the implementation project, except maintenance and support of the standard software, is done by someone else, including for example configuration, application operation.

To avoid monopoly for complex projects where external expertise on running the projects with the chosen system, it is advantageous to select a product vendor that has a network of partners familiar with the system, who can act as product integrators. This point is less important than avoiding monopoly on the daily operation, as large complex projects following the main replacement typically are limited in number.

The approach outlined does not remove vendor lock-in; the view taken here is that such a thing is not possible. However, most of the serious issues normally associated with vendor lock-in can be avoided.

6.8 Use of standard systems

Using a standard system is a comprehensive outsourcing of the IT required for executing substantial amounts of the functions that make a telecommunication business function. The basic rationale for using standard systems is that the potential differentiation from building systems internally is dwarfed by the cost, risk and complexity of building and maintaining such systems for a single company.

The standard systems typically contain a substantial functionality that covers not only what the business currently needs, but also requirements that are not recognized at the time of installation.

Standard systems, used appropriately, also promotes simplicity and standardization of the transaction processes.

Use of custom solutions are viable in selected areas where the value of flexibility and speed can justify the higher complexity and cost of custom systems.[7]

Standard systems continuously come in new versions that give new functionality in line with the "best practices" of the industry. And finally, they provide cost predictability.

There are some downsides and caveats also. The most important downside is that it can be a very challenging task to understand the application and utilize it in the best way. Understanding the application is a requirement for being able to operate it (and failure to secure this capability will over time remove commercial leverage for the customer). Further, in a project driven by a product configurator with focus on deliverables, nuances of requirements may be missed causing a necessary re-implementation after the project when the system capabilities and limitations have been fully understood.

Handling the issue of understanding the application is discussed further under the discussion of knowledge transfer, section 7.8. The questions of what constitutes a standard system and whether standard systems are always a viable approach are discussed below.

6.8.1 Definition of a standard system

The issue of what constitutes a standard system is the focus of this subsection. This is not a straightforward question, and one that is dependent upon the specific system. It includes:

1. The ability to upgrade seamlessly. When a system is non-standard, upgrading it to new versions or even handling new database or security standards, can be a major undertaking.
2. Amount of configuration, irrespective of which form it takes, in order to fulfil the functional requirements.
3. Implementation of required configuration with "pure configuration", i.e., not coding even if such coding can survive new versions.

Ten-twenty years ago, "standard" was a fairly simple concept. Configuration was a set of fixed entries in a table or a file that defined the behaviour of a system, e.g., currency conversion rates or the interval before starting a dunning flow. Such configuration items still exist, but for some systems scripting a rule engine or even software code can survive an upgrade without any intervention; for SaaS applications such an ability is frequently a requirement.

[7] These considerations refer to the normal situation of network operators. For organizations where a small team, say less than 20 people, can develop and maintain the entire system stack, custom development can be a viable strategy.

Since the benefits of standard system, apart from its functional abilities, is closely linked to the ability to do seamless upgrades, the concept of a standard system becomes a legal rather than a technical item. The following definition is offered here as a starting point: A "standard system" has the following characteristics:

1. The functionality it provides is well documented.
2. It has regular updates in the form of releases with release notes, explaining what the upgrade is and how it is applied.
3. Applying a release takes place using automated scripts and a reasonable release test with predictable cost. Not an upgrade project per se.
4. It has firm rules for what is *permissible customizations*, i.e., changes or configurations that may be applied without losing the advantage of being able to upgrade using scripts delivered by the vendor and basic verification only.
5. The system is constructed in such a manner that the advantages of the system may be obtained in real-life operations within the limits of permissible customizations.

With the complexity of modern systems, the concept of "standard system" must be understood in context of the specific technology of potential systems. The above is a starting point, but configuration may still become so complex that it can be a maintenance nightmare.

In order to qualify as a standard system, the vendor should be willing to put this in a contract, including taking responsibility (and liability) for upholding the result. This way the problem of defining what 'standard' means in strictly technical terms goes away and is transferred to a legal requirement.[8]

Being a standard system is not, however, enough. The system should support the required business functionality with only "nominal" configuration. Now, what "nominal" is and how it is measured is specific for each system and this needs to be evaluated as part of the RFP.

6.8.2 Suitability of standard systems

The point of view taken here is that for telecom operators, standard systems are suitable for all the core transactional processes. That does not imply that their footprint should be universal. There can certainly be areas where customized solutions can make sense. One obvious candidate is online, i.e., the systems that end-customers interact with directly, where frequent changes and tests of customer behaviour requires fast changes and full flexibility. Another area is the still-explorative area of machine learning and AI.

Again, in the view taken here, there are two key requirements for using custom systems:

1. It is a conscious decision based on real business benefits derived from flexibility and control of the system.
2. It is clearly contained architecturally (so that for instance the online solution does not start making its own price calculations independently of the main product catalogue).

6.9 Project roles

For a greenfield implementation project, there are a number of roles involved. Clarity on the roles is important for the sourcing process, since filling the roles are part of what the process secures. The following taxonomy is used here.

Product vendor is the supplier of the actual software, including the subsequent support and maintenance services. For SaaS applications, the product vendor also supplies the subsequent operations of the system.

Product configurator is the supplier of the configuration of the software. This role includes soliciting the detailed requirements, starting from the business requirements and use cases.[9] The product configurator should have a product-specific process with templates and other tools for defining the detailed requirements and be able to staff the work with a team that have experience from several similar projects. The product configurator also builds the main outgoing interfaces, e.g., direct debit, printing, provisioning, and secures that relevant APIs are made available, e.g., self-service. Further, the product configurator executes the test of the implemented product permitting it to be part of the final end-to-end acceptance test and participates in the end-to-end test. Finally, the product configurator has a role in migration, at least for loading data into the new software.

System integrator is the responsible for managing the entire implementation project. This can be handled internally, using a specialized firm and/or individuals and is typically a combination. The system integrator manages the end-to-end project, including the dependencies to other components, securing all is ready for end-to-end acceptance, building an end-to-end test model, securing that

[8] Note that a database system is a standard system in this definition. Which is fine, but fairly uninteresting for the purpose of implementing business support systems.

[9] The business requirements and use cases are artifacts developed through the process outlined in this document, see further below.

the parts of migration not delivered by the product configuration are implemented. Organisational change management is also part of this role.

In the process suggested here, the best approach is to have the product vendor and the product configurator be the same company. The main reason for this is that modern software products are so complex in functionality and configuration that only people with deep and extensive experience in the specific product can manage the process well.

The system integrator can be the same company as the product vendor and product configurator. However, few of the product vendors have this competence, and even those who have, are reluctant to take on the full role. The execution ability should in this case be reviewed carefully.

The typical dilemma in selecting vendors for the roles is that the product vendor is typically very strong as product configurator but less capable as system integrator. Classical, general integrators (e.g., Accenture, TCS, CapGemini) typically are strong as system integrators but not as qualified as product configurator (with a few exceptions on systems where they have large practices). Finally, the general integrators can find the role of system integrator without the product configurator role a bit thin to assign senior staff to.

6.10 Invite system integrators?

This section discusses the question of whether a system integrator should be involved. A wish to have a system integrator to "deal with the problem" is not uncommon. This section discusses this point a bit further through two scenarios.

The **first scenario** is where the new IT system exclusively or materially is delivered by the product vendor who also takes the product configurator role.

In this case, the system integrator role is (optionally) to run the process outlined here by helping to define and document the architecture, write the requirements and use cases, draft contracts, preparing for execution, etc., essentially extending the customer staff, supplying the system integrator role. The system integrator role can continue into implementation, still representing the customer side. This part is difficult to manage commercially and tends to be time and material. It is, however, less of an issue than the other elements since it has no role after the project and the proportion of cost in this role is quite low, typically less than 5% of the total project cost.

An alternative is to run the sourcing process internally (or with dedicated advisors) and invite system integrators to run the overall project management after the product vendor contract is in place.

Part of the challenge in this scenario is to find a system integrator willing to take this comparatively small role.

The **second scenario** is where the system integrator has responsibility for the entire solution, adhering to the normal industry use of the "system integrator" term. Here, the tender would go towards a system integrator who would partner with a software vendor to deliver the solution. In case of multiple software vendors, e.g., operational CRM and billing in separate systems, the role of the system integrator becomes very prominent.

In this case the system integrator will have a commercial interest in moving away from the standard system since it will increase the work outside and therefore the revenue of the system integrator. Most system integrators with substantial implementation practices have sales incentives that will emphasize this. Therefore, using a system integrator this way will require strong oversight internally.

A further key disadvantage of having a system integrator in a very prominent role is that, for telecommunications, the system integrators rarely have sufficient insight in the software to deliver it in a good way. This means that the ability to optimize overall value creation may be severely compromised. This is contrary to many ERP solutions (e.g., SAP, D365) where the delivery model is based on system integrators who have staff with deep system insights.

In summary, the preferred model here is to invite product vendors to the tender, expect them to be product configurators also and use other externals for system integrators.

6.11 Identifying vendors

Prior to executing the contracting phase, the list of vendors to invite must be produced. Several sources exist for this, with the analysis firms like Gartner and Forrester being the most prominent. For the very initial round, it is better to spread the net rather widely. Through setting requirements on the process fairly strict, including that only vendors with complete functional footprint and relevant references are considered, a level of self-selection can be achieved.

6.12 Methodology

In defining the methodology, the pitfalls outlined above should be avoided. In addition to these, the design principles applied here are:

1. Optimize overall value creation through focus on "what" and utilize vendor expertise to define "how", jointly defining a good fit to the standard product.
2. Maintain commercial leverage throughout and after the implementation.

The real key point here is to optimize value creation. If one insists on specific solutions and processes, it will not necessarily fit into the systems proposed. The product vendors will know their systems so well that they can propose the best way of achieving the business targets – if one lets them have the freedom to do so. This is one of the places where the use of system integrators replacing the software vendor can be challenging.

The suggested approach is outlined below, showing the process once the gross list of vendors has been found.

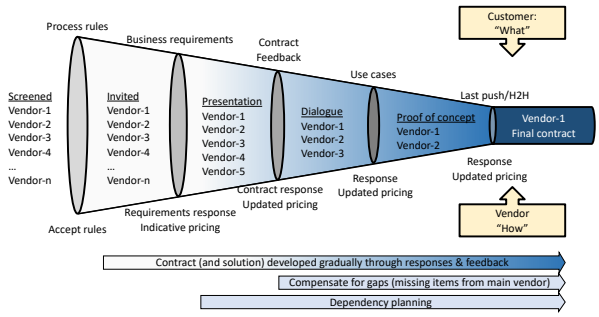

Figure 14: procurement process

The process looks like a classical procurement funnel: Many vendors enter the funnel, and one eventually emerges as the selected. It also resembles it a lot, but it also has important differences; it is designed to develop the solution jointly with the product vendor through the sourcing process.

In the illustration, the customer actions are at the top and the vendor responses at the bottom. So, for instance at the start, the customer publishes the "process rules" and the vendors "accept rules" (assuming, of course, that they actually do accept the rules).

The funnel is shown as a strict waterfall with one ring following the previous. When converting the method to a plan, this is not the reality. Developing the business requirements, the term sheet (or contract, see section 6.12.3 for a discussion of the contracting approach) and the use cases are time-consuming activities and should be started in parallel.

At each ring of the funnel, the vendors can be evaluated and sorted. The process is flexible with respect to the number of participants except for the last phase where the number of vendors should be down to two.

The steps "business requirements" and "term sheet" can be exchanged, depending upon the readiness of material for the phases. They can also be combined, but that takes out one option to push vendors for better terms.

6.12.1 Process rules

The first step is to invite vendors including setting out a set of process rules. The process rules should explain how the process works, including:

1. Description of the process, timeline, contacts.
2. Rules governing the communication, e.g., who it is permitted to speak with, process for questions.
3. Rules governing the negotiation process, e.g., whether 'second round' bidding is permitted.
4. Any response during the process shall become part of the final agreement. This includes, in particular, that minutes and recording of all presentations commit the vendors and become part of the final agreement.
5. Adherence to the principle of partial specification (see section 6.3.5). This is truly crucial as it permits outsourcing, at least in part, of the "fit for purpose" risk.

6.12.2 Business requirements

The second step is to develop and publish the business requirements. The business requirements describe the business to be supported, including products, processes, number of customers per segment, compliance, security, technical environment into which the solution must fit, etc. This is typically a document of 50-200 pages, and it is core to the process as it represents the foundation of what needs to be delivered. The document, therefore, typically needs approval by a broad number of stakeholders. The document should be developed by a small core team to secure consistency and concurrently validated with relevant stakeholders. The vendors respond to the document with a statement of compliance and a non-binding price indication.

The business requirement development relies heavily on the principle of partial specification. While it still makes sense to cover the main customer facing processes, configuration items, and products, the business requirements can be kept at the indicated extent due to the principle of partial specification. In its absence, the business requirements would need to be a lot more extensive.

The greenfield approach to some extent builds a "new company" on the side of the old one and moves the customers to that. This, of course, is an exaggeration since the brand, primary

infrastructure and all processes not involving customer facing transactions are unchanged. Nonetheless, it does typically represent a very significant renewal of the business.

Writing the business requirements assumes that a view of future business exists or can be developed in the process. As such, a simplification is to some extent a pre-requisite for starting work on the business requirements.

This does not mean that every detail of the simplification is developed prior to the implementation phase. Part of the process outlined is to develop the more detailed requirements jointly with the vendor. This means that one needs to know where the business is going – the "what" part in Figure 14. Segments, brands, products, volumes, territories, governance, overall application architecture – these are things that need to be understood when drafting the business requirements. But other parts of the simplification are provided through the process of joint specification in adherence to a standard system.

When defining new or updated processes, understanding the situation from the customer perspective, often termed the "customer journey" is crucial. This implies, for instance, keeping the customer perspective and testing against real customers and incorporating their feedback. Also important is to maintain a process close to reality; not infrequently, customer journey visions are not viable in practical operations. The customer journey definition should be developed hand in hand with the process definitions – they really are two sides of the same coin. Customer journey perspective helps keeping customer experience in focus. Process definitions help keeping practical operation as the necessary basis. This relationship is particularly important when employing consultants that only focuses on one part – in itself that may be fine as not all have both competences, but the development should still be closely co-ordinated.

6.12.2.1 Flexibility requirements

As most transformation projects are born from an extended period of irritation over inability to launch market initiatives at a fast pace due to complexity of legacy processes and IT, a typical high-priority item in the requirement list is flexibility.

Flexibility obviously is important, but it is important to be specific as to how this is understood and interpreted. Requiring vague, broad, general flexibility, e.g., "ability to support future business models, marketplaces and partnering" will always yield a "compliant" answer from all vendors and be of little practical value.

6.12.3 Contracting basis

The third step is contracting basis. There are a couple of options for this.

Obviously, one can merely accept the vendor's standard contract and negotiate from there in the final step. That, generally, is commercially challenging.

To avoid this, two different approaches can be applied: writing and submitting a full contract for the vendors to consider or requesting adherence to key demands listed in a term sheet.

Writing and submitting a full contract is the most thorough approach and secures that all pertinent aspects are covered before final down-selection. The main disadvantage is the effort required in writing the contract and negotiating several different mark-ups. The approach assumes a high-quality contract being submitted – otherwise the contracting team will be quality assuring in parallel with multiple vendors, an almost impossible task.

A lighter approach is to submit a term sheet that covers the items that are normally contended in software contract and ask for compliance. This will then be incorporated in the contractual material. Before submitting the term sheet, the vendor should also be asked to provide the standard contract so additional terms may be added. The advantage of the process is that it requires significantly fewer resources. The key disadvantage is that it leaves certain negotiations until the final contracting, where the commercial leverage is weakened.

Jointly with the contracting input, the vendors are given feedback on their response, both the content and the price. The vendors then respond to the contracting basis and reverts with updated pricing as well as updates to the response to the business requirements, if applicable.

6.12.4 Use cases

The fourth step is issuance of use cases the vendors must support. The purpose is to ensure a structured walkthrough of key functionality and capabilities that may be documented as a contractual commitment.

Any reasonable functional description technique may be applied – use cases is one option that has the advantage of being fairly easy and efficient to apply.

The functional descriptions need not be fully covering all requirements, but should as a minimum cover the core functionality. The vendors shall respond with compliance to the use cases, preferably including a description of how the use cases are supported, and potentially submit updated pricing.

Also, the use cases form the basis for the proof of concept.

6.12.5 Proof of concept

Before entering the fifth step, the number of vendors should be low, preferably down to two. There are two main reasons for this. Firstly, it is quite a bit of effort for the sourcing team to go through a proof of concept. Secondly, the vendors will need to put significant effort in a proof of concept, and their awareness that their chances are good will improve the quality.

In this step, the vendors demonstrate how their solution will support the business requirements and the use cases. This should be done through a "proof of concept workshop" of 4-8 days where the relevant use cases structure the walkthrough and the compliance is recorded, electronically as well as in minutes.

For this activity, it can be considered to permit a limited payment to the product vendors entering into the proof of concept. This goes in particular for smaller contracts where the vendor sales management may be reluctant to spend substantial time of configuration without a contract. This approach is akin to the paid "architects' competition" employed in some tenders for construction.

6.12.6 Contracting

Finally, a vendor is selected, and the concluding negotiations can take place. Potentially, the project can start signature based on a letter of intent or similar. This, of course, carries the risk that the project is abandoned, but with the proper construct, it serves to put some of the time pressure on the vendor. If the LOI approach is to be applied, it should be included in the process rules.

6.12.7 Organisation

For this phase the project organisation should be very simple, preferably only roles rather than streams. A very common approach is to have disconnected streams, e.g., 'technical stream", "functional stream", "sourcing stream", "legal stream" etc. This approach carries the risk of getting a disconnected process; for instance, the business requirements are not independent from the contract and getting functionality into the standard product may be a more important task for the sourcing staff than the final price reduction.

The approach recommended here is to have a small, closely-knit team with roles but not independent streams. And a project management team (or individual if such a person can be found) who has the ability of taking a holistic view of the documents produced.

Ideally, the core team should be fewer than 10. Additional people for review should be added for quality assurance and securing buy-in for the process. These additional people should be focused on providing review, inputs on questions, and providing business guidance, but not producing material.

6.12.8 Staffing

Staffing the business side of a transformation project is almost invariably a challenge. For an approach as outlined above, the issue becomes paramount.

As discussed above, the approach outlined in this document attempts to define a "new company". This entails a very large number of micro-decisions on what are acceptable compromises to support the chosen segments. These decisions will have significant impact on the enterprise value following the transformation as it may shift market share and revenues beyond normal evolutionary development of business.

Therefore, the requirements cannot be specified and approved in "steering group approval" style, as is the normal mode of involving senior decision makers. It requires that these senior decision makers are directly involved in these micro-decisions; not CxO level, but experienced people 1-2 layers below.

Freeing up such people from daily operation is generally impossible. They are invariably indispensable to delivering the results in the next few quarters.

One potential solution to this is to strengthen these line organisations several months before the project is to start, so that the senior people in effect have been backfilled and can be freed up for the project.

The focus in this section has been on the business staff; that does not mean that IT staff is irrelevant or unimportant, but typically IT staff is organized to work in projects, making it easier to allocate them. Furthermore, the IT staff is substantially augmented by the vendors, in particular the product configurator and system integrator.[10]

6.12.9 Other preparations

There may be other solution components than the main software system. In addition, there are typically dependencies on internal and external systems; for instance, a test environment for the direct

[10] See section 6.9 for a discussion of these vendor roles.

debit or number porting systems may not be readily available.

In *Figure 14: procurement process*, these activites are illustrated in the arrows below the main funnel termed "compensate for gaps (missing items from main vendor)" and "dependency planning".

In parallel with the sourcing process (preferably integrated in the same team), these dependencies and gaps must be understood and detailed. This can entail running smaller scale sourcing processes, securing priority in internal resource allocations, reaching agreement with existing, external suppliers for legacy systems that are to be retained, securing development of dual operations and other similar activities.

It is not uncommon to run a full process of sourcing for product vendor, product configurator, and system integrator, and complete the development and signing of a contract - only to find that the internal readiness is not up to supporting the process for which a contract has been signed. The product vendors, recognizing this, can leverage such situations to avoid living up to obligations, getting waivers for penalties, or similar actions.

7 Greenfield phase 2: implementation

The implementation phase can obviously be executed in a number of different ways with differing approaches for the pertinent activities, e.g., development, testing, project management. The approach described here continues from the results coming out of phase 1 and has been tried in practice in various flavours.

The section starts with a methodology discussion followed by a review of a sample project organisation. Not that the organisation structure is really that important, but jointly with the methodology, it serves as a framework for discussing the various roles and responsibilities.

Following this, a number of key implementation considerations are discussed:

1. People and change.
2. Dependency management.
3. Test.
4. Migration.
5. Knowledge transfer.

This list is not exhaustive – obviously there is a lot more to running an implementation project of the scale discussed in this comparatively brief document than can be covered here.

7.1 Methodology and plan

As will be recalled from the above, the approach addresses a situation where a vendor – either a product configurator or a system integrator – delivers very large proportions of the project. Other deliverables, e.g., interfaces, adaptations to existing systems or new online presence, are delivered in the project or by other vendors, internal or external. All this needs to come together in order to secure the delivery. In addition, in order to keep cost control, the functional requirements need detailing prior to execution. Therefore, the overall program approach resembles an old-fashioned waterfall.

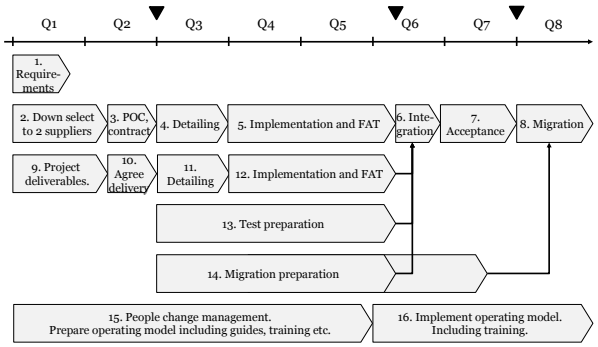

Figure 15: generic implementation plan

The shaping phase are in Q1 and Q2 and the implementation phase from Q3 onwards with testing happening in Q6 and Q7 and migration following that.

Note that the waterfall approach is identified via the key milestones indicated: contracting, start of test and start of migration. Here, all the components need co-ordination in time and content. How each delivery is managed, via agile or waterfall, is not material to the process.

A normal reaction is that this is a long time. Two years, of course, is that. However, the timeline outlined is an optimal one requiring extremely efficient execution. Three years is a more common execution time. It is an almost invariable experience that pushing the timeline too hard will result in delays that in the end makes the project take even longer.

The activities 1-3 are described in the contracting phase, section 6 above.

Activities 9 (project deliverables) and 10 (agree delivery) identify and secure delivery of such items that the main process (activities 1-8) are dependent upon, e.g., preparation of dual operation, data cleansing. The dependencies can be managed by the project or can be delivered by the organisation

or an external (typically not the vendors for the main system). Irrespective of who and how, agreement of delivery must be in place in order to execute at the time indicated.

Activities 4 (detailing) and 5 (implementation and FAT) will depend on the approach from the product configurator. The detailing will normally be required as the specification developed in the contracting phase will not be complete. At the end of activity 5 (implementation and FAT), the product configurator should deliver a solution that has been through internal testing (the FAT, i.e., factory acceptance test).

Activities 11 (detailing) and 12 (implementation and FAT) correspond to 4 and 5 and focus on implementation of the dependencies, typically related to temporary interfaces or interfaces to existing systems that remain after the project. Here the approach can vary quite a bit, and for internal deliverables more flexibility on end-result is possible. For instance, the online presence can be more or less sophisticated in an initial release and that can be managed flexibly internally. Similar to the deliveries from the vendor out of activity 4 and 5, the deliveries must be tested and functioning internally.

Activity 13 (test preparation) is preparation of test which is further described in section 7.4. It prepares test of integration, migration, security, disaster recovery, acceptance, manual processes, operations etc. Note that the start of the activity is early – preparing a proper test model is extensive work requiring on-going interaction with the detailing and implementation.

Activity 14 (migration preparation) is preparation of migration. Like test, it starts early as there is substantial work in designing dual operations, identifying data sources and initiating data cleansing. Migration is further discussed in section 7.7.

Activity 15 (people change management) and 16 (implement operating model) are the preparation and implementation of the required change. See further discussion in section 7.3.

Activity 6 (integration) is the integration where the various elements meet "officially". Clearly, it is advantageous if integration between the various systems has taken place prior to this activity in order to enable on-going learning, but this is a detailed planning issue not covered further here. The integration testing is a key milestone, the latest time at which all systems meet and a precondition for meaningfully entering acceptance.

Activity 7 (acceptance) is the execution of acceptance procedures where the prepared test scripts are executed in order to verify readiness for go-live. For vendors, in particular the vendors for the main system, it also has a contractual dimension.

Activity 8 (migration) is the physical migration. See further in section 7.7.

7.2 Project organisation

In the contracting phase described in section 6, the project organisation is small and with little or no formal structure, as further discussed in section 6.12.7. The process in the first phase is explorative and knowledge sharing is key.

Moving to implementation, more structure is required. In the implementation phase up until start of integration, a typical organisation structure is as illustrated below in *Figure 16: generic implementation organisation*. The discussion in this section assumes the use of a product configurator, whereas a system integrator can be used as augmentation of internal staff.

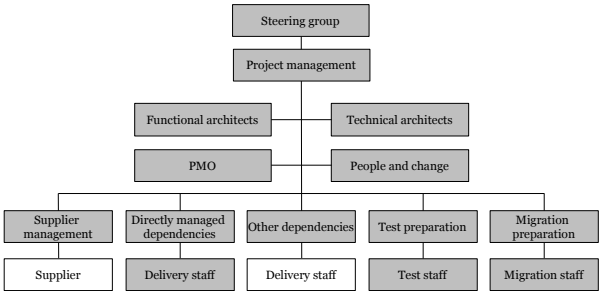

Figure 16: generic implementation organisation

The structure is typically adjusted in the test phase to accommodate the focus on error correction. In case of multiple releases, corresponding adjustments are required. The white boxes are dependencies that are not managed by the project. The grey boxes are project roles that can be internal or staffed from a system integrator or other means of staff augmentation.

The project management is a combined content and progress role, with primary responsibility of securing overall business goals.

The functional architects are key people designing the future business. They are responsible for formulating the business requirements and the further detailing. They should be mandated to define the future business, i.e., to take all the detailed decisions that are required in order to design a new IT system, even when it is based on a standard system. Through a reference group or other structures, suitable involvement and approvals in the line organisation can take place.

The functional architect is often the most difficult role to fill since it requires knowledge of the current business, a vision of the future and sufficient seniority to balance the vision with the realities of project execution. Such individuals are invariably already in key roles; if it does not hurt severely when taking them out of the line organisation, they are likely to be the wrong people.

The technical architects are responsible for securing alignment with technical standards, interfaces, overall architecture, issue resolution etc. They write the technical parts of the business requirements and detail the interfaces, data mastering and other similar items. If preparation of operations is included, infrastructure architects design and secure this.

Typically, a fairly large number of complex issues needs to be resolved, requiring both function, business understanding, insight in current systems and ability to analyse and facilitate agreement on technical solutions. While typically simpler to staff than functional architects, these are also highly specialized roles.

The PMO is the administrative part of project management, handling tasks like resource allocation, financial follow-up, physical accommodation. This is more generic role.

The people and change are responsible for driving the change within the line organisation. Again, this is fairly generic, but preferably the change competence should be combined with people understanding the organisation well.

Vendor management secures link to the product vendor and product configurator and their deliverables, managing the contract, resolving issues, arguing about cost and potential change requests etc.

Directly managed dependencies is a subproject that handles the technical deliverables that are directly from the project, i.e., handled with staff allocated to the project.

Other dependencies task is a subproject that handles deliverables from other parts of the organisation, other external vendors and any such dependencies that can reasonably be pooled together.

The three last roles, the vendor management, the directly managed dependencies and other dependencies, require project management and business understanding, interacting with the functional and technical architects. They can be structured in different ways. Here it is illustrated based on relationship with the programs, but it could also be a functional delineation, e.g., one part responsible for the channel part of the systems or value stream based.

Test preparation is the team that prepares the test model. In some cases, the software vendor does this, and then some of the staff would not be directly managed.

Migration preparation is the team that prepares for data migration, data cleansing and other similar activities. Securing dual operation can also be placed here.

Most of the management roles described above are part of the system integrator role. These can be staffed internally, from a specialized firm, through sourcing of individuals or a combination of these approaches.

7.3 People and change

In section 2 the main organisational change challenge of mobilizing the entire organisation towards the transformation goals was discussed.

At a more pedestrian level, people and change is concerned with preparing the organisation for the changes resulting from introducing the new tools. The new tools may imply changing roles, e.g., higher first-time resolution will shift staff from back-office to front-office. And they will certainly require updates to guides to front-end staff, training, information meetings and a number of similar tasks.

Securing that these activities all take place is the role of people and change. The methods and approach for such activities are a comprehensive topic in its own right and not covered in this document.

The point of this section is to highlight the importance of agreeing distribution of responsibilities. The experience from the projects that form the basis for this document is that the project itself should be focused on orchestrating the change. The execution of the change activities should take place in the line organisation. This includes for instance:

1. Preparing and implementing change to the organisation structure, if applicable.
2. Developing process and instruction material for use of the new systems.
3. Preparing and executing training.
4. Updating KPIs, department and individual targets etc.

The main reason is that unless the line organisation takes responsibilities for these activities, the project becomes something that is done *to* them rather than *with* them, a perception which tends to create opposition and friction

7.4 Dependency management

From the system integrator perspective, the key focus is to manage dependencies towards the points in time where all the various items come together.

The individual project streams must manage their dependencies directly, such dependencies to be identified and monitored on an on-going basis.

A key dependency is that all sub-projects must be ready for integration testing, i.e., the place where the components all come together for initial end-to-end testing. Managing this well typically requires some progress tracking.

7.5 Contingency management

As noted initially, even with the best preparation by experienced people, transformation projects tend to have significant uncertainties. For this reason, a contingency is often allocated to handle the unexpected.

The contingency can be explicit and allocated or implicit as a risk. Having it explicit and allocated in the budget ensures that it is not cut out as a saving without substantial consideration. The contingency should be seen as part of the project budget, since it reflects the professional experience on what must be expected in a well-executed project. It is not, as the perception sometimes is, a result of sloppy planning.

Further, it can be managed at different levels. Either at the project level or at steering group level. The best approach is for the project to administrate it, since that is the practical execution, but for the steering group to manage (approve) its use, since this promotes discipline.

The most important point is to make the contingency explicit and distinguishable from the core project budget in order that it is not merely used without specific decision to do so.

7.6 Test

Test happens at many levels in major system replacement projects. Each of the deliverables that jointly constitute the solution have internal tests, interface tests, process tests etc.

The topic of this section is the process test that verifies that all the functionality is in place. A very light version of it constitutes the integration test that precedes the process test. From the V-model perspective, these are the top 1-2 layers of testing; the deeper layers are handled in the individual sub-streams.

Once the process test has been executed successfully, the solution is ready to go live and be accepted. Securing that this takes place with acceptable risk is the – very important – role of the end-to-end process test.

In addition, the end-to-end test is also linked to formal acceptance and typically also payment milestones.

Testing this way resembles classical system test in a waterfall project. However, the effort compared to "classical" projects is reduced from two factors. Firstly, large parts of the solution are based on standard systems, so a lot of basic errors will be eliminated even before start of the project. Secondly, the migration approach is assumed to be gradual, which means that the tolerance for errors is substantially higher than in big bang migrations. This risk balance is an important determinator for the complexity and duration of the test.

In order to execute the test, a test model must be in place. A test model consists of a number of test cases, typically 1,000 – 2,000, that are joined into scripts for execution. Part of the test model is kept long-term for regression test. Once the test model and the system are stabilized, it may be considered automating the test execution.

The test model typically consists of a large number of functional test cases and fewer, but crucial, security and compliance test cases, including business continuity, penetration test and the like. These are registered in a collaboration tool that supports the fixing, deployment and retesting process, including registration of execution, errors, fixes, re-testing.

In order to execute the test, there needs to be comprehensive environments spanning all relevant systems. A number of such environments are typically required in order to execute integration test, acceptance test, technical tests, training etc. Depending upon the maturity and sourcing model of IT, this can be anywhere from straightforward to extremely challenging.

Furthermore, data is required. In these days of GDPR, data can either be constructed or anonymized. Both options can entail significant effort.

Since the product vendor in many cases builds substantial parts of the test model, it is tempting to use this for acceptance test. That is also quite viable, but firstly the vendor normally only covers part of the scope and secondly the acceptance test also has a commercial implication in signifying the completion of the vendor's commitments in the project. As long as these issues are addressed, it can save substantial amount of work.

Various levels of test automation are beneficial and should be mandatory for the lower-level tests like unit test as well as for areas that lend themselves to

batch or API testing. Setting up test for GUI can be quite time consuming and must be considered against the benefits (seeing that only a fairly small proportion is likely to be used after the transformation project has concluded).

Equally important as automating test of code is the management, again preferably automated, of test environments. It is a common and very time-consuming challenge that the migration between environments, code deployment, data synchronization, connectivity between the intended environments etc., is of insufficient quality. This can, for example, cause problems to appear that are not due to the application. Differences between test environments and production environments can cause applications that passed acceptance in test to fail in production.

7.7 Migration

Migration of customers to a new system can take place gradually over a longer period, or over just a couple of days, the latter termed 'big bang'. The approach outlined in this document assumes gradual migration. The key reason is the risk associated with big bang migrations, where faults can cause customer operation to halt, ultimately threatening existence of the company. In order to avoid such risks, a big bang approach requires a more extensive test than what is indicated in the plans presented above.

The gradual migration assumes dual operations, i.e., the parallel execution of the legacy and the new system for a period of months. During this timeline, a number of areas need to interface, including for example online, call centre PBX, payment interfaces, CDRs, number porting. In addition, the resources like phone numbers, SIM cards, CPEs, and IP addresses need to be managed jointly. Some resources can be pre-allocated to either stack, but others like physical addresses and access ports must be shared.

The dual operation needs to be prepared and preferably put into production well before the go-live of the solution (to avoid confusing where the errors are coming from).

The migration can be driven by the customers, in which case dual operation is a year-long affair, or through moving customers actively from one system to the other. In the latter case, the tempo needs to be adjustable in order to mitigate the risk; this means that changes made to the customer cannot imply that the customer migration date becomes fixed (like it might be if the customer should have notice of the change).

Technical migration can be product based or customer based. In the product-based approach, all customers with product X are moved to product Y. This is analytically simple and often used. In case of many high-ARPU customers moved to lower-ARPU products it can also be expensive. Another option is to make it customer-based, defining the new product based on customer profile and usage. This is more complex but can in some cases mitigate the ARPU decline, since the customers can be migrated to "value loaded" products, i.e., getting a product with more consumption or features rather than a decrease in price. The customer-based migration may require customer notification, which then shifts the risk balance of the migration.

Migration is nearly always challenging. Data is typically inconsistent and/or incorrect, data sources are hard to identify, and dual operation adds to complexity. The preparation should be started early with actual data and the migrated data should be used for testing as soon as possible. As the saying goes, if migration is not your biggest problem you have not understood it yet.

7.8 Knowledge transfer

As was noted in section 5, getting the right target operating model is important to secure continued commercial leverage. In order to enable a target operating model where the product vendor does not continue to deliver services except for support and maintenance, other staff, internal or external, needs competence.

The competence includes application operation, application maintenance and application development. The latter two in particular include ability to configure the system as well as interfaces.

Due to the complexity of the systems contemplated here, such competence cannot be achieved fully through training. People need to work with the systems to understand their potential and shortfalls, selecting and advising on the best way to use the system. This is best accomplished through having people participating in the development jointly with the vendor. If this is not possible, it must be expected that the vendor needs to participate in all activities for a while following go-live. Firstly, this should be budgeted and secondly, participation of internal staff in the activities following go-live should be enforced.

Knowledge transfer can be elusive and tend to suffer from focus on more urgent activities. One potential remedy is to make it subject to formal acceptance.

8 Summary

The greenfield approach outlined here provides a process for implementing a new core system. The process has been tried in practice, adapted for individual projects to different flavours.

The first phase, contracting, is the most important since it secures the simplification and adherence to standard systems. In particular, for the simplification, strict adherence to standard systems is important. And, equally important, it tees off for the implementation in a manner where expectations are aligned.

The second phase is more regular in the sense that it resembles other approaches to system implementation.

While not applicable to all contexts and still requiring substantial experience to execute, it contains a framework that make such projects relatively predictable.

As stated initially, it is a key purpose of the process to ensure long-term cost predictability. This is secured through the following measures:

1. Strict adherence to standard systems.
2. Contracting for fixed-cost implementation.
3. Securing ongoing commercial leverage in all areas except support and maintenance.
4. Contracting for cost predictability in the support and maintenance.
5. Securing termination notices that do not undermine cost predictability for support and maintenance.

This, of course, provides a *basis* for cost predictability. If the subsequent governance permit deviation from the standard system, it will be a question of time before one return to the "legacy" problem. While the people and change part of the project can focus on this, it remains a management task and challenge to keep adhering to simplicity and standard systems.

9 Contact

The authors of this document are Lars R. Andersen and Simon Skals. Lars and Simon work as advisors, primarily within networks and IT.

Lars has a background from consulting, primarily from Accenture, and CTO/CIO of Telenor in Denmark.

Simon has a background from the telecom industry, having worked in companies such as Telenor and TDC, and was co-founder and CITO of the Danish ISP Hiper.

We can be contacted via: lars@ra-advisory.dk.
This and other white papers are available free on www.ra-advisory.dk.

The document is written from experience with a number of BSS replacement projects and gradually refined to reflect the learnings obtained. In most of these projects, the methods have been applied in part and in some they have been used successfully in full. Now clearly, there is a lot more to BSS replacement than what can be contained in this document. If you wish further perspectives please feel free to reach out.