POE-: Towards an engineering framework for solving change problems

Journal Item

POE-Δ: Towards an engineering framework for solving change problems

Georgi Markov, Jon G. Hall, and Lucia Rapanotti

Department of Computing and Communications
The Open University, UK
{*georgi.markov*, *jon.hall*, *lucia.rapanotti*}@*open.ac.uk*

Abstract. Many organisational problems are addressed through change and re-engineering of existing Information Systems rather than radical new design. In the face of up to 70% IT project failure, devising effective ways to tackle this type of change remains an open challenge. The paper discusses the motivation, theoretical foundation, characteristics and evaluation of a novel framework - referred to as POE-Δ, which is rooted in design and engineering and is aimed at providing systematic support for representing, structuring and exploring change problems of socio-technical nature. We generalise an existing theory of greenfield design as problem solving for application to change problems, using a design science research methodology. From a theoretical perspective POE-Δ is a subset of its parent framework, allowing the seamless integration of greenfield and brownfield design to tackle change problems. Its initial case study evaluation, consisting in its application to a real-world change problem of realistic complexity, shows that POE-Δ allows the systematic analysis of change problems, leading to clearer understanding and more informed decision making.

Keywords: organisational change, information systems, software engineering, change problems, problem solving, design theory

1 Introduction

Continuous change is part of the make-up of the modern organisation, and crucial to its survival and success in an ever-increasing competitive and volitive business environment. Information systems are often at the heart of organisational change, either as drivers or enablers, in their essential role within complex organisational socio-technical systems. This has led to the introduction of a multitude of approaches and theories on how to approach organisational change successfully. Yet a large proportion of change initiatives end in failure [1]: while the reasons are many and complex, some have been attributed to a lack of systematic guidance and direction, particularly in key re-design steps which are part of the change process [2–4]. To remedy this deficiency, some have advocated the need for a structured, detailed design theory from which detailed methods can be derived to guide the change process [5–7]. The work presented in this paper makes a contribution towards addressing this challenge.

Our overall research aim is the definition of a framework for addressing change problems of a socio-technical nature systematically, by providing the means to represent, structure and explore those problems. The framework is rooted in design and engineering, and extends an existing design theoretic framework for problem solving called Problem Oriented Engineering (POE in short, [8]), hence its name *POE-Δ*. The paper discusses the motivation, theoretical foundation, characteristics and early evaluation of POE-Δ.

From a methodological perspective we have taken a design science approach [9], starting from an initial theoretical proposition based on POE, followed by empirical evaluation in the form of a case study in the context of a real-world organisation.

The remainder of this paper is organised as follows. Section 2 discusses some background literature. Section 3 introduces POE-Δ and details of its initial evaluation. A discussion is given in 5, while 6 concludes the paper.

2 Background

2.1 Organisational change

In the field of Management Science organisational change is defined as "the process of continually renewing an organisation's direction, structure, and capabilities to serve the ever-changing needs of external and internal customers" [10]. It has been the subject of study since the early 1950s, resulting in a multitude of approaches and theories on how it can be successfully tackled (see, e.g., reviews in [11, 12]). While there are significant differences between these approaches, a common criticism is that they only provide very high level change process descriptions to guide the organisation through change initiatives, and by and large are rather imprecise and unmethodical, and that this is true of even influential approaches adopted in practice, such as Total Quality Management (TQM) and Business Process Reengineering (BPR) [13]. Specific criticisms include: a lack of "a systematic approach that can lead a process redesigner through a series of steps for the achievement of process redesign" [2]; a lack of "actual technical direction to (re)design a business process" [3]; a limitation to "descriptions of the 'situation before' and the 'situation after', giving very little information on the redesign process itself" [4].

Many have advocated the need for *design thinking* [14] in organisational change, and indeed this has become increasingly topical, with many articles discussing its role as an enabling concept which can be embedded within a development approach to organisational change (see, e.g., [15, 16]). However, still lacking is a design theory and systematic methods to make the re-design process more structured, easier to operationalise and less dependent on the creativity and intuition of single individuals, a clear objective advocated by some [5, 6]. This is where our research comes in.

2.2 Design Problem Solving

Design problems were defined by [17] as real-world problems which are ill-stuctured and complex. Ill-structuredness implies that the starting point is often vague goals and intentions, with unclear success criteria, hence many degrees of freedom in the problem statement and no unique path to solution [18]. Complexity refers to number of issues and variables involved and their relationships, but also to their stability over time and uncertainty which is beyond the problem solver's grasp or control [19]: as such complexity also relate to how difficult a problem is to solve for a human problem solver.

With its origin in engineering and system sciences, the notion of design problem has been extended to any design artefact, whether a physical product or a system or just a course of action [20]. In this sense, some have argued that many organisational problems [21, 22] can be seen as design problems.

Design problem solving [20] has been explored widely in the context of creating a new artefact, whether a physical product or a system or a course of action [20], so-called *greenfield* design. Design problem solving for *brownfield* design, that is design aimed at changing a current situation in order to meet some new need in context, remains under-explored. With our research we contribute a systematisation of design problem solving in the case of change problems.

2.3 POE

POE, defined by the second and third authors, is an engineering framework with an accumulated body of work spanning over a decade, including application and evaluation through a number of real-world engineering case studies. Its underlying design theory concerns the characterisation of individual problems and how problems relate and transform to other problems as part of problem solving processes. A thorough presentation of POE is beyond the scope of this paper, but can be found in [24]. Here, we briefly recall some basic definitions, extended by POE-Δ to change problems in Section 3.

A POE problem is "a stakeholder's recognised need in context." For stakeholder G, with recognised need N_G in real world context E_G, we define their problem to be the pair:

$$(E_G, N_G)$$

E_G and N_G are to be understood only as place holders, as G's initial conceptualisation of their problem may have neither solution nor sense. Irrespective of sense or solution existence, G's wish becomes a challenge to designer D to make sense of G's problem by finding an agreed environment E and need N, leading to D's problem

$$E(S) \Vdash_G N$$

which reads "Find S which, when installed in E, meets N to the satisfaction of G."

D's challenge consists of all the solving problems activities that lead to the solution of G's problem. Someway through problem solving we encounter D's

variously detailed E, N and S and form a judgement as to whether a problem has been solved. We do this by creating a solution for it through a sequence of judgement-preserving transformations, i.e., transformations that the relevant stakeholders would agree preserved solvability, that move a problem to known solved problems. Thus, a problem is solved if and only if it can be transformed to known solved problems. As part of the transformation sequence, a solution to the problem is created.

Treatment of the physical world in POE is based on the work of Jackson and others [25–27], which relies on the notion of real-world phenomena and their relationships. For the purpose of this discussion, a phenomenon can be simply characterised as "an element of what we can observe in the world." Hence a problem environment E is a set of *domains* $[D_1, ..., D_n]$, each a set of related *phenomena* that are usefully treated as a unit for the purpose of problem solving (c.f., [26, Page 270]). Domains interact through their sharing of phenomena; behaviourally, a domain maps a collection of phenomena to a timeline of their occurrences and interactions [27].

The POE problem solving process proceeds through a set of identifiable steps, which are instances of recurrent problem transformation types, e.g. sensemaking [28]. These are defined as "software problem transformation schema," each describing a general way in which a problem may be related to other problem(s). While POE transformation schema are beyond the scope of this paper, in Section 3 we will see examples in the context of POE-Δ.

3 POE-Δ

POE-Δ shares and extends a number of POE's characteristics, including: elements of its semantics; its graphical notations; and its underlying process pattern [23]. However, while POE deals with 'greenfield' development, POE-Δ deal with change, or 'brownfield,' problems which are solved not solely by the design of a new artefact, but by a change of, and addition to, existing artefacts within a target context (a system, product, process, etc). As such POE-Δ introduces a number of significant extensions to both the core concepts behind the original framework and to the formal and graphical notations, as well as to the underlying process pattern, while however always remaining compatible to the core POE.

POE-Δ's aim is to support users in systematically analysing and solving change problems by helping them to better represent, structure, and explore them to pinpoint where intervention is necessary and what the ramification of that intervention could be.

Like POE, POE-Δ is not supported by computational tools. Rather POE-Δ links current tools and notations in new ways to support the change engineer.

In the following we briefly introduce formal elements of POE-Δ and report on its initial evaluation trough an organisational case study.

3.1 Change Problems

The inspiration behind POE is Rogers [29] definition of *engineering* as:

> the practice of organizing the design and construction of any artifice which transforms the physical world around us to meet some recognized need

Rogers appeared to have had in mind greenfield engineering; indeed, POE focusses on the production of the *artifice*. In contrast, POE-Δ focuses on the *transformation* of the *environment before* E into an *environment after* F which will meet the need.

POE-Δ imports POE's notions of need and environment, defined in terms of the domains located therein, and their phenomena, and the domains that share phenomena can interact.

Like POE, POE-Δ prescribes no single description language for a problem's elements; indeed, different elements can be described in different languages.

In defining change problems, we begin from the same place as POE[1]: we suppose that change problem owner G recognises a need in the real world and wishes that need to be satisfied. From G's perspective, then, a problem P is a pair, consisting of a real world context E_G and a need N_G. We make no assumption that G's view of the real world context is realistic or representable, nor that G's recognised need has a solution.

Irrespective of sense or solution existence, we will assume that G's wish becomes a challenge to change engineer D to make sense of G's change problem (E_G, N_G) and to solve it. D's challenge thus consists of (cf. [24]):

CPS1. creating their own view, (E, N), of the G's change problem (E_G, N_G);
CPS2. receiving validation from G that (E, N) is properly representative;
CPS3. identifying a new environment F consisting of i) those the parts of E that can remain unchanged, together with ii) changes to E, and iii) any additions necessary to effect the change;
CPS4. receiving validation from G that F meets the agreed recognised need N;
CPS5. migrating from E to F.

Like POE [24], even if expressed linearly as bullet points, the challenge facing D may be iterative and highly non-linear.

Given the above, we consider a change problem to be a four-tuple (cf., [24]), written

$$E \Delta F \Vdash_G N$$

where E and F are both collections of domains and N is a need.

Unlike E which is arrived at from an understanding of the real world, F is a designed object, created through iteration of the steps CPS3 and CPS4. As such, and without constraining the languages in which existing and new domains are

[1] The presentation loosely follows [24]

or will be expressed, it is constrained in the forms it can take which are defined by the following grammar:

$$\begin{aligned} \mathit{ChangeDomain} &:= \mathit{Domain} \mid \mathit{Domain}[\{\mathit{Changeable}\}](\{\mathit{NewDomain}\}) \\ \mathit{Changeable} &:= \mathit{ChangeDomain} \mid \mathit{CP} \mid \cancel{\mathit{Domain}} \mid \mathit{Domain} \triangleright \mathit{NewDomain} \\ \mathit{Domain}, \mathit{NewDomain}, \mathit{Need} &:= \mathit{Name} \end{aligned}$$

There are a number of representational conventions in the above grammar (illustrated in Figure 1). Suppose the hierarchical domain $C = [D, E, F, G]$. Then:

1. $(\{\})$ is written $()$;
2. in the first clause of the *Change Domain* production, domain C is undecorated. This identifies C as a changeable domain, i.e., its embedded domains will be subject to further change analysis;
3. in the second clause, given C, we have an expression such as $C[E, \cancel{F}, G \triangleright H](J)$, by which we mean: (i) D (elided) remains unchanged; (ii) E is changeable (cf. the previous clause above); (iii) F will be removed by the change; (iv) G will be replaced by H (which, thus, observes and controls the same phenomena as G); and (v) J will be introduced by the change.

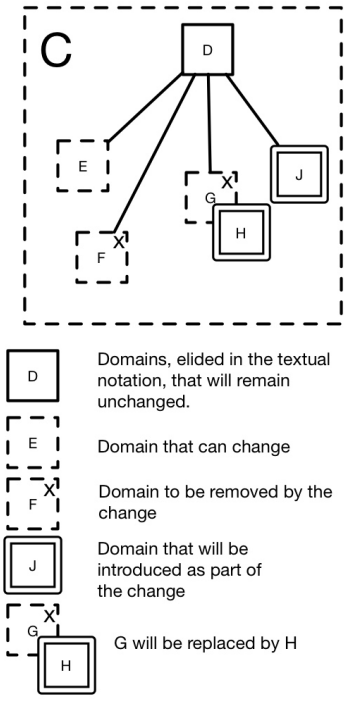

Fig. 1. The *ChangeDomain* notational conventions

A change problem is said to be *greenfield* whenever each *Changeable* element is either $\cancel{\mathit{Domain}}$ or $\mathit{Domain} \triangleright \mathit{Domain}$, i.e., each changeable element is either deleted or replaced. A change problem that isn't greenfield is *brownfield*. Although we do not define the equivalence here, a greenfield change problem is equivalent to a (*tangle* of) POE problem(s) [24]. As we shall see, as well as forging the link with POE, the notion of a greenfield problem is at the core of what we mean for a change problem to be solvable.

Example 1. A software house's CEO, G, is in reflective mood and wishes to reduce the technical debt in *ColourMaker* (shortly *CoMa*), a commercial image manipulation app. Georgi, an experienced software engineer, is tasked with doing so.

Following Step CPS1 (perhaps in collaboration with G) Georgi captures the initial change problem in POE-Δ as:

$$[\mathit{Swift}, \mathit{CoMa}]\Delta F \Vdash_G \mathit{RTDNeed} \tag{1}$$

in which [*Swift*, *CoMa*] is Georgi's understanding of the problem owner's change problem context, i.e., the *CoMa* app and its *Swift* environment; *RTDNeed* is his understanding of their need; and F stands for the new environment that Georgi will create to solve the problem.

Assuming that the problem owner is willing to validate Georgi's initial change problem (Step CPS2), focus can turn to analysis of the changes needed to solve the problems by creating the new context F. If validation was not forthcoming, Georgi would iterate his understanding until either i) he gained validation, or ii) problem solving was ceased.

3.2 Change problem transformation

Following POE, we define change design as the step-wise transformation of change problems. Suppose, then, we have change problems $E\Delta F \Vdash_G$, $E_i\Delta F_i \Vdash_{G_i} N_i$, $i = 1 \ldots n$, $(n \geq 0)$ and *step rationale* J, then we write:

$$\frac{E_1\Delta F_1 \Vdash_{G_1} N_1 \qquad \ldots \qquad E_n\Delta F_n \Vdash_{G_n} N_n}{E\Delta F \Vdash_G N} \langle\langle \text{J} \rangle\rangle$$

to mean that $E\Delta F \Vdash_G N$ is solved with *rationale* $(AA_1 \wedge \cdots \wedge AA_n) \wedge J$ whenever $E_i\Delta F_i \Vdash_{G_i} N_i$, $i = 1 \ldots n$, $(n \geq 0)$ are solved with rationale $AA_1, \ldots, AA_i$ respectively.

Below the line is the conclusion problem; above the line are the premise problems. Steps can be combined to produce entire *change development trees*.

In POE-Δ, we establish that a change problem is solved when it can be transformed to a collection of solvable greenfield problems via the above transformation.

As in POE, there are many classes of transformation, which we do not have the space to present fully. As an example, one such transformation is SUBSTITUTION:

Substitution Given a substitution $PS = [A_i/E_i]$ of change problem elements and step rationale J_{PS}, we have the substitution step:

$$\frac{P[PS]}{P} \langle\langle \text{SUBSTITUTION } J_{PS}\text{; environment: } E[PS]\text{, need: } N[PS]\text{, change: } F[PS] \rangle\rangle$$

which capture the situation when, if the appropriate stakeholder is satisfied that the substitution is correct, then the original problem will satisfy the stakeholder when the substitution is made.

A substitution represents, for us, an increase in our knowledge[2] of the problem element substituted, as illustrated in the following example.

[2] Technically, knowledge is validated belief, with validation arriving from the problem owner. It is thus a stakeholder-relative notion.

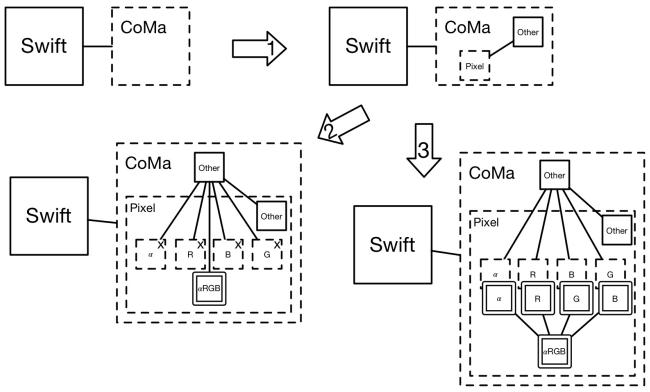

Fig. 2. Illustrating Georgi's change problem; see Example 2

Example 2. Georgi begins his analysis of the changes needed (Step CPS3). Initially, he identifies that the *Swift* environment won't need to change in satisfying *RTDNeed*, but that *CoMa* will. Georgi's first change problem transformation is to apply the substitution $Subs_1 = [F/CoMa]$ giving

$$[Swift, CoMa]\Delta CoMa \Vdash_G RTDNeed \tag{2}$$

This is the starting point in Figure 2.

Next, through a detailed code inspection of *CoMa*, Georgi recognises that the *Pixel* class's getters `getRedValue`, `getGreenValue`, `getBlueValue`, and `getAlphaValue` perform the same steps and share significant code, each being distinguished only by their use of different constants in the method bodies. He recognises a recurring software engineering problem for which a best practice remedy is available in the *parameterise method design pattern* [30] which prescribes the combination of similar methods through a switch parameter. Applying this design pattern will reduce technical debt. Georgi therefore applies the SUBSTITUTION transformation with the substitution $Subs_2 = [CoMa/CoMa[Pixel]()]$ indicating that the *Pixel* component will be subject of change, whilst leaving the remainder of *CoMa* unchanged (Other in the Figure), giving:

$$[Swift, CoMa]\Delta CoMa[Pixel]() \Vdash_G RTDNeed \tag{3}$$

which is Step 1 in Figure 2.

Initially, Georgi considers reducing technical debt through the substitution[3] $Subs_3 = [Pixel/Pixel[\not{R},\not{G},\not{B},\not{\alpha}](\alpha RGB)]$ which would leave (Step 2):

$$[Swift, CoMa]\Delta CoMa[Pixel[\not{R},\not{G},\not{B},\not{\alpha},](\alpha RGB)]() \Vdash_G RTDNeed$$

but then realises that, as αRGB has a different signature to each of R, G, B and α, such a change cannot be achieved with changes local to *Pixel*, i.e., he would

[3] For brevity, we abbreviate the getters to R, G, B, and α, respectively.

need to consider change to the whole of *CoMa* contradicting the assumption underpinning Equation 2 that only *Pixel* would change[4].

Thus, Georgi rejects this choice, backtracking to Equation 2 and, instead, reconsiders the substitution $Subs_3$ from which he derives $Subs_4 = [Pixel/Pixel[R \triangleright R', G \triangleright G', B \triangleright B', \alpha \triangleright \alpha'](\alpha RGB)]$, in which the getters are replaced rather than removed, leaving (Step 3):

$$[Swift, CoMa]\Delta CoMa[Pixel[R \triangleright R', G \triangleright G', B \triangleright B', \alpha \triangleright \alpha'](\alpha RGB)]() \Vdash_G RTDNeed$$

As all changeable domains are either new or replaced, this is now a greenfield problem, for which a solution can be sought for in POE.

Concatenating the steps Georgi took gives the following change design tree:

$$\dfrac{\dfrac{\dfrac{[Swift, CoMa]\Delta CoMa[Pixel[R \triangleright R', G \triangleright G', B \triangleright B', \alpha \triangleright \alpha'](\alpha RGB)]() \Vdash_G RTDNeed}{[Swift, CoMa]\Delta CoMa[Pixel]() \Vdash_G RTDNeed}\ \langle\langle \text{SUBSTITUTION}, Subs_4 \rangle\rangle}{[Swift, CoMa]\Delta CoMa \Vdash_G RTDNeed}\ \langle\langle \text{SUBSTITUTION}, Subs_2 \rangle\rangle}{[Swift, CoMa]\Delta F \Vdash_G RTDNeed}\ \langle\langle \text{SUBSTITUTION}, Subs_1 \rangle\rangle$$

4 Case Study Evaluation

We conducted an initial evaluation of POE-Δ via a case study, which was set up as a live project which took place in the context of a multi-national organisation, as contribution to a work package in an EU funded research project[5]. The goal of the work package was to showcase the adoption of the emerging Open Services for Lifecycle Collaboration[6] (OSLC) standard in a real-life industrial setting with production-grade tools. The case study organisation led the work package and provided the tool, a model-based test design editor and generator, for which adoption of the standard was to be realised. Also, as OSLC is intended to enable integration and interoperability between software tools from different tool vendors, the work package required the involvement of, and where appropriate knowledge transfer to, other third-party tool vendors, including at least one SME.

The whole project took nine months from conception to completion. Five people were involved in the case study, working in Europe and India. The first author, as principle investigator for the work package, led the POE-Δ case study work. POE-Δ was used to identify both technical and organisational changes and to design any new and replacement artefacts needed.

[4] Of course, this assumption might be incorrect, but that could only be established in consultation with G at Step CPS4.

[5] *https://www.eitdigital.eu/innovation-entrepreneurship/cyber-physical-systems/*

[6] *http://openservices.net*

4.1 Objectives

The main evaluation objectives were to collect early evidence of whether POE-Δ is:

- sufficiently expressive for application to a real-world change problem;
- able to support the systematisation of the problem solving process; and
- able to cope with real-world complexity.

Moreover, we were interested in assessing its performance as a means of communication among project stakeholders, as well as its ability to surface critical design issues to be resolved.

4.2 Procedures

At the start of the project the first author introduced POE-Δ to relevant stakeholders within the organisation, including the project manager, tool architect and several developers. While a formal development was recorded by the first author throughout, only problem descriptions based on the graphical notation (similar, but not identical to, that used in Figure 2) were shared among project stakeholders. In particular, the notation was used for communication in most technical meetings, but was not used for the upstream communication to project management and external stakeholders.

The problem solving steps outlined in Section 3 were followed, with iteration applied as needed.

A debriefing with stakeholders, in particular the technical stakeholders who had the most exposure to the framework, was held at the end of the study, in the form of one-to-one conversations, to gather some qualitative feedback on the overall performance of POE-Δ in the project.

4.3 Main problem solving steps and related artefacts

An initial need, in the form of three high level requirements, R_1 to R_3 (not reproduced here), was initially articulated by the case-study organisation as problem owner. The initial steps focused on exploring those requirements and gaining understanding of the relevant change problem context (Step CPS1). This involved acquiring knowledge and information by informally consulting a number of stakeholders and analysing the available tool and development documentation on processes, architecture and organisational setup, as well as the available OSLC specifications in order to understand their specific technological requirements. Through this initial knowledge acquisition step, the PI was able better to understand the problem and its domain, and introduce enough structure into it, in order to be able to capture it as the following POE-Δ problem:

$$P_1 : [MBT\_Tool, DevOrg_{MBT\_Tool}, OSLC\_standard, extTools]\Delta F \\ \Vdash_{Case-Study\_Org.} R_{1.1 \wedge 1.2 \wedge 1.3 \wedge 1.4 \wedge 2.1 \wedge 2.2 \wedge 2.3 \wedge 2.4 \wedge 2.5 \wedge 2.6 \wedge 3.1 \wedge 3.2 \wedge 3.3}$$

where:

- $R_{1.1}$ to $R_{1.4}$ were derived from the original R_1 through the analysis of the OSLC specification and the therein contained technological constraints (i.e. adoption of a RESTful Programming Model).
- $R_{2.1}$ to $R_{2.6}$ were derived from the original R_2 through the analysis of the architecture of the host organisation's model-based tool.
- $R_{3.1}$ to $R_{3.3}$ were derived from the original R_3 through several interviews with various team members and team leads in order to better understand what functions will be migrated to the new offshore team.
- MBT-Tool is a domain representing the host organisation's own model-based tool with all its relevant components.
- $DevOrg_{MBT\_Tool}$ is a domain representing the development organisation behind the MBT-Tool, including teams, roles, skills and responsibilities.
- OSLC standard is a domain representing the OSLC standard with its various specifications.
- extTools is a domain representing the third party (from the point of view of the host organisation) tools involved in the already described workflow (Figure 3).

When complete the change problem was validated by the problem owner (Step CPS2).

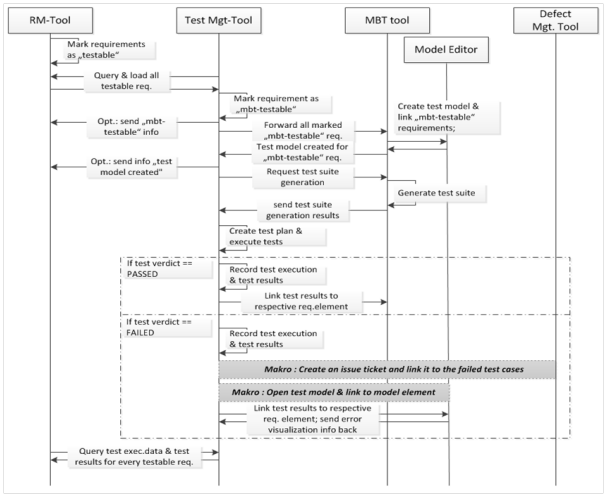

Fig. 3. The to-be-supported workflow using OSLC

The analysis proceeded with several alternating steps of problem, i.e., environment and need exploration (Step CPS1) and validation of findings by the relevant stakeholders (Steps CPS), until all relevant domains in the problem context and the need were revealed. At this stage of the analysis the problem environment included 27 domains.

This high problem complexity made the change analysis steps (CPS3) very challenging. Due to the many dependencies between exposed environment domains, several of the change artefacts had to be *co-designed*[7], i.e., changes in one domain would have an immediate impact on another domain. The colour coding in Figure 4 indicated the complex overlaps between domains.

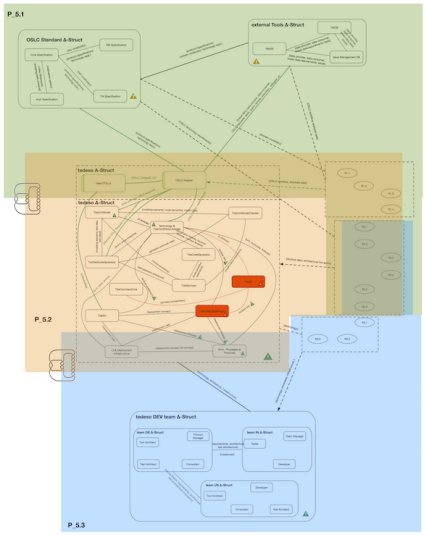

Fig. 4. "The many dependencies between exposed environment domains": green corresponds to external tools/OSLC; blue to the Development organisation; red/orange to the MBT-Tool.

Therefore our next step involved isolating the domains on which the tangle depends, design the necessary changes for the remaining domains, and finally reintroducing the changed domains and in a greatly simplified context re-attempt the co-design of the tangled domains as shown in Figure 5. This resulted in a successful solution, which during the validation (Step CPS4) was accepted by all stakeholders, including the relevant EU institutions.

That is where the study stopped. Step CPS5 — migrating from original environment E to the newly defined F — is still ongoing within the work package.

5 Discussion

In the case study, the application of POE-Δ allowed us to make good progress towards understanding and structuring the initial problem situation, to a point

[7] POE calls problems that share phenomena in this way tangled problems. Tangled problems are not always susceptible to simple divide and conquer problem solving strategies.

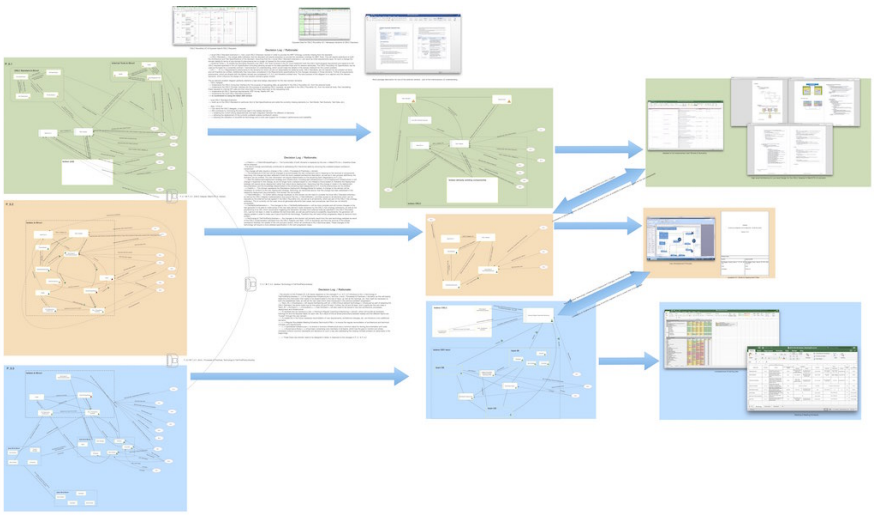

Fig. 5. Handling co-design

where the problem could be understood by all involved stakeholders, and elements in its context which needed to change could be identified. Further, it provided step-by-step guidance during the change design phase, allowing one to identify clearly: (i) domains that will need to be modified by the solution; (ii) domains that will indirectly influence or be influenced by it or as a result of its introduction to the organisation; and (iii) domains that will remain untouched by the change.

The analysis ended with greenfield problems, to be solved within POE. In fact, the migration between POE and POE-Δ and back was seamless, thanks to the shared theoretical basis of the two frameworks. It was noted that the framework provides effective tools to focus on specific change subproblems, while still allowing the consideration of co-design issues which are the result of the ways various problem tangle. As such it provides both the means for focussed analysis while reducing the risk of ignoring some important dependences.

The way the case-study organisation usually handles this type of change problem varies depending on the kind and source of change, but mostly involves some combination of strategic planning, architecture and change management, with high reliance on experts. It was felt by stakeholders that the POE-Δ approach taken in this case study was able to bring about results comparable to those which might have been obtained through the usual company approach, while increasing the potential for repeatability and reducing reliance on experts. Whether this is actually the case, will be the subject of future studies.

The evaluation also provided some initial findings in regards to the POE-Δ notation, which was used as a communication medium in most technical meetings. At this technical level, the graphical notation was well understood and its adoption was straightforward. The feedback from the stakeholders was that the

use of the graphical notation, instead of its formal counterpart, in the problem analysis steps was crucial to increase the acceptability and learnability of the framework. To avoid confusion however it seems important to note, that the POE-Δ graphical notation is not a modeling language, and its goal is not to provide a mean for functional decomposition of the system. Its main goal is instead to help make structures and relations more explicit and, thus, support understanding, communication and the discovery of new relationships. This according to the participants was an important factor for the fact that the framework was so well received as to be adopted also in other projects.

The main matter of concern brought up by most stakeholders was the lack of a suitable software tool keep to track of problem models and their relationships, to validate the consistency of problem transformation steps, and provide heuristics or even automation for some intermediate design steps.

6 Conclusion

This paper has introduced POE-Δ, a framework for design as problem solving which provides systematic support for representing, structuring and exploring change problems. The paper has discussed its initial formalisation and an early evaluation via a case study concerning technical change in the context of a multi-national organisation, involving a number of stakeholders. The findings are encouraging, with the framework performing reasonably well in its first full-time application in an industrial setting. The case study work has provided some initial evidence that POE-Δ is expressive enough for application to a real-world change problem, and able to support the systematisation and semi-formalisation of its problem solving process. In such a way POE-Δ can help to reduce ambiguities, explicate tacit assumptions, and to a certain degree to objectify, consistencify and as such standardize the change process, which in turn will reduce its current strong dependence on human expertise, intuition, and creativity.

Current research aims at a full formalisation of the framework, and its further evaluation in industrial practice. Issues of scalability, repeatability and tool support will be given priority.

References

1. B. Burnes and P. Jackson, "Success and failure in organizational change: An exploration of the role of values," *Journal of Change Management*, vol. 11, no. 2, pp. 133–162, 2011.
2. G. Valiris and M. Glykas, "Critical review of existing bpr methodologies," *Business Process Management Journal*, vol. 5, no. 1, pp. 65–86, 1999.
3. H. Reijers and S. L. Mansar, "Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics," *Omega*, vol. 33, no. 4, pp. 283 – 306, 2005.
4. H. Gerrits, "Business modeling based on logistics to support business process reengineering," in *Proceedings of the IFIP TC8 Open Conference on Business Process*

Re-engineering: Information Systems Opportunities and Challenges, (New York, NY, USA), pp. 279–288, Elsevier Science Inc., 1994.
5. M. J. Verkerk, *Trust and Power on the Shop Floor: An Ethnographical, Ethical and Philosophical Study on Responsible Behaviour in Industrial Organisations*. Eburon, 2004.
6. A. Kleiner, "Revisiting Reengineering," *strategy+business*, no. Third Quarter 2000 / Issue 20, 2000.
7. R. T. By, "Organisational change management: A critical review," *Journal of Change Management*, vol. 5, no. 4, pp. 369–380, 2005.
8. J. G. Hall and L. Rapanotti, "Software engineering as the design theoretic transformation of software problems," *Innovations in Systems and Software Engineering*, vol. 8, no. 3, pp. 175–193, 2012.
9. A. Hevner and S. Chatterjee, "Design Research in Information Systems: Theory and Practice," 2010.
10. J. W. Moran and B. K. Brightman, "Leading organizational change," *Journal of Workplace Learning*, vol. 12, pp. 66–74, Jan. 2000.
11. A. H. van de Ven and M. S. Poole, "Explaining development and change in organizations," *The Academy of Management Review*, vol. 20, pp. 510–540, July 1995.
12. A. M. Pettigrew, R. W. Woodman, and K. S. Cameron, "Studying organizational change and development: Challenges for future research," *Academy of management journal*, vol. 44, no. 4, pp. 697–713, 2001.
13. G. Cao, S. Clarke, and B. Lehaney, "The need for a systemic approach to change management—a case study," *Systemic Practice and Action Research*, 2004.
14. P. G. Rowe, *Design thinking*. MIT press, 1991.
15. M. Eneberg and L. Svengren Holm, "Design thinking and organizational development: twin concepts enabling a reintroduction of democratic values in organizational change," EAD - European Academy of Design, 2013.
16. A. Deserti and F. Rizzo, "Design and organizational change in the public sector," *Design Management Journal*, vol. 9, no. 1, pp. 85–97, 2014.
17. D. H. Jonassen, "Toward a design theory of problem solving," *Educational technology research and development*, vol. 48, no. 4, pp. 63–85, 2000.
18. H. A. Simon, "The structure of ill-structured problems," in *Models of discovery*, pp. 304–325, Springer, 1977.
19. J. Funke, "Solving complex problems: Exploration and control of complex systems," *Complex problem solving: Principles and mechanisms*, pp. 185–222, 1991.
20. G. F. Smith and G. J. Browne, "Conceptual foundations of design problem solving," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 23, no. 5, pp. 1209–1219, 1993.
21. R. L. Martin, *The design of business: why design thinking is the next competitive advantage*. Harvard Business Press, 2009.
22. B. Leavy, "Design thinking–a new mental model of value innovation," *Strategy & leadership*, vol. 38, no. 3, pp. 5–14, 2010.
23. J. G. Hall and L. Rapanotti, "Assurance-driven design in problem oriented engineering," *International Journal on Advances in Systems and Measurements*, vol. 2, October, 26-31 2009. http://oro.open.ac.uk/19123/.
24. J. G. Hall and L. Rapanotti, "A design theory for software engineering," Technical Report TR2016/01, Department of Computing and Communications, The Open University, Walton Hall, Milton Keynes, MK7 6AA, 2016.
25. C. Gunter, E. Gunter, M. Jackson, and P. Zave, "A reference model for requirements and specifications," *Software, IEEE*, vol. 17, pp. 37–43, May 2000.

26. M. A. Jackson, *Problem Frames: Analyzing and Structuring Software Development Problem*. Addison-Wesley Publishing Company, 1st ed., 2001.
27. J. G. Hall and L. Rapanotti, "A reference model for requirements engineering," in *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International*, pp. 181–187, IEEE, 2003.
28. K. E. Weick, *Sensemaking in organizations*, vol. 3. Sage, 1995.
29. G. Rogers, *The nature of engineering: a philosophy of technology*. Macmillan Press, 1983.
30. *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.