BizDevOps Blueprint: A Lightweight Model Supporting Web Application Design by BizDevOps Teams

Web applications (WebApps) are becoming more complex, as users and businesses increase their needs and demands. At the same time, WebApp development is becoming more sophisticated, in particular with the emergence of BizDevOps, which helps organizations to accelerate and better align software delivery with business needs. In this landscape, hard modeling approaches have started showing their limitations, held by extensive analysis and formalization, while developers must deal with looseness and volatility. This study suggests the adoption of lightweight modeling. It contributes a blueprint which offers a simple, flexible strategic map for BizDevOps teams to design WebApps. The blueprint was subject to both formative and summative evaluations, including interviews with experts. The results indicate the blueprint can be useful in providing an overview of WebApps projects.

Keywords *- Lightweight modeling, blueprint, web application design, BizDevOps.*

Introduction

Modern web applications (WebApps) are expected to support rich user experiences, including interactivity, flexibility and personalization. This requires the adoption of a variety of most used but often dreaded web technologies (Stack Overflow, 2022). At the same time, developers are hard-pressed to design, develop and deploy software continuously, often within time frames of less than one hour (Gearset, 2022). This requires new paradigms like BizDevOps. The main goal of BizDevOps is to bring Biz (business development), Dev (software development) and Ops (operational deployment) expertise together to accelerate and better align software delivery with business needs (Lohrasbinasab *et al.*, 2020).

These expectations lead us to question the role of modeling in WebApp design. For several decades, modeling has had a central role in systems design across the information systems and computer science

fields. Arguably, most developers with a university degree in these fields will have been taught "traditional" modeling (e.g., entity-relationship diagrams, UML) for data, entities and processes (The Joint ACM/AIS IS2020 Task Force, 2021).

However, traditional modeling does not seem to resonate with contemporary discussions around WebApps. First, many modern WebApps break down functionality into a myriad of loosely coupled pieces. Such pieces can be loosely represented (e.g., using application programming interfaces, APIs for short), but not using traditional models, because they entail structure, coherence, sequencing, and rigor (Bucchiarone, Dragoni, *et al.*, 2020). Second, many modern WebApps privilege highly volatile, distributed events over stable, centralized data, thus making it difficult to model system behavior (Baresi and Garriga, 2020). There are a very large number of possible paths through modern WebApps. Third and finally, continuous re-design turns models into highly ephemeral objects that can become outdated within hours. Considering such looseness, volatility and ephemerality, teams have few incentives to model the traditional ways.

Even though several lightweight modeling approaches exist, none supports the highly dynamic and varied settings and practices required by BizDevOps teams (Chasioti, 2019; Ebert *et al.*, 2016). Existing approaches still privilege an upfront definition of requirements and features, and have weak links to the all-hands-on-deck, integrated practices adopted by BizDevOps (Alt *et al.*, 2021). For instance, they make it difficult to break a WebApp into micro-applications, which is a key enabler of continuous software delivery (Dudjak and Martinović, 2020).

Considering this context, the research problem addressed by this study is *how to elaborate a lightweight modeling approach that supports WebApp design by BizDevOps teams*. Lightweight modeling offers a holistic and simplified approach to WebApp design. It also offers an agile and "innovation friendly" yet strategic approach to design and enhance WebApps, aiming to build teams' consensus rather than formalizing and detailing ephemera.

This study proposes a *blueprint supporting WebApp design by BizDevOps teams*. The term 'blueprint' refers to a script defining the strategic, high-level control of a system (Camilli *et al.*, 2017). The proposed blueprint should offer a lightweight approach to address the limitations of existing approaches, that is:

1) organize a set of concepts used by modern WebApps; 2) tailor and align these concepts to rapidly define WebApps at the strategic level; 3) support an agile, "fast fail" mind-set, while still enabling a disciplined approach to managing complexity; and 4) support a holistic view of a WebApp, which integrates Biz, Dev and Ops.

The rest of this paper is organized as follows. First, it reviews the theoretical background to the blueprint. Then, it defines the requirements of the approach. This is followed by a description of the components and an illustration. The evaluation of the blueprint is presented. Finally, the paper offers a discussion and conclusion.

Background

From traditional to lightweight models

Since traditional modeling is not effective in supporting contemporary discussions around BizDevOps, a new approach is required. The BizDevOps paradigm seeks to empower teams to adopt their own approaches, rather than being governed by a standardized corporate approach (Lohrasbinasab *et al.*, 2020). As such, it can be difficult to agree on a long term set of models for BizDevOps.

On the other hand, a general trend seems to be emerging in many business and technical environments, which consists in adopting lightweight models. This is driven by the desire of many organizations for increased agility and continuous innovation. The trend has been observed in various fields more or less related to WebApps. For instance, a variety of "canvas" models have been proposed to represent business and technological innovations (Antunes and Tate, 2022a). Other examples include the use of personas, storyboards, empathy maps, and customer journeys in systems design (Wood *et al.*, 2021). All these approaches avoid using overly specialized and detailed rules, and intricate formal languages. Instead, they provide holistic and simplified views of key elements of systems. Lightweight models are associated with a culture of "fast fail". They offer enough detail to allow ideas to be developed, discarded, modified, and refined, without investing a large amount of time in analyzing and preparing detailed models of an idea that contains significant flaws (Tate *et al.*, 2018).

Modeling epistemology

To discuss the characteristics of models required for BizDevOps, it is necessary to begin with a basic understanding of models and their epistemology. "A model is an abstraction of some system. The system can be a man-made system […] or a natural system. […] Models of systems are used for various purposes such as for developing system theories, understanding a system's behavior, predicting some system outcome, designing a new system, or modifying an existing system. We note that there may be different models of the same system as models can be developed for different purposes. Usually, a parsimonious model of a system is desired, meaning the model is as simple as possible yet meets its purpose. Also, the accuracy of a model required is only what is needed to satisfy its use or purpose. Models are commonly used […] instead of the system itself because often experiments cannot be performed on the system (e.g., if the system does not exist) or it is too costly to experiment on the system." (Sargent, 2015 our emphasis).

In the philosophy of science, models can be understood as a representation of the world (consistent with the definition above), and modeling as a form of "representational practice" (Giere, 2004). Following Giere (2004), we can characterize modeling using a set of relationships with the following form:

S uses M to represent W for purposes P

S is the subject creating a model *M* using a set of symbols. *W* is the system the subject aims to represent. *P* concerns the specific purposes leading the subject to create *M*. Most modeling approaches related to systems development seek to create *M* that provide *faithful* representations of *W* (Burton-Jones *et al.*, 2017; Burton-Jones and Grange, 2013). This has been underpinned in representation theory, with its particular focus on faithfully modeling the concepts, states and state-changes of *W*, which are considered the core aspects of a system (Recker *et al.*, 2019). The term 'faithful' underlines the argument that useful *W* require faithful (i.e., accurate and complete) *M*, otherwise the users of *W* would not be able to perceive the meaning of *W*, and conversely, *S* would not be able to properly design *W* (Burton-Jones *et al.*, 2017).

However, another element in this set of relationships is *P*, and by moving attention to *P*, we may end up relaxing the common assumptions about the nature of *M*, in particular regarding accuracy and completeness (Alter and Bork, 2019). In fact, various purposes *P* may lead developers to relax

faithfulness. For instance, faithfulness may be impossible or inadequate in the early design stages, where certainty and information quality are low (Samset and Volden, 2016). Excessive attention to faithfulness at an early stage may hamper ideation, innovation and strategic thinking (Alter and Bork, 2019; Erickson *et al.*, 2005). Thinking centered on faithful representation, including (for example) developing accurate and complete accounts of system components, increases cognitive load (Alter and Bork, 2019). Further, it also leads developers to digress from the epistemic qualities of a system, i.e., meaning and function, towards less crucial concerns (Knuuttila, 2011).

Monolithic modeling should also be avoided to increase the agility and flexibility required by BizDevOps. Achieving greater agility and flexibility has been a major motivation for a shift to more agile software development paradigms (Sarker *et al.*, 2009; Siau *et al.*, 2022). Considering the potential influences of *P* on how *M* represents *W*, two categories of modeling approaches are considered, designated as hard and lightweight modeling. Next, the two approaches are further discussed.

Hard modeling approaches

Hard modeling approaches adopt an epistemologically realist position, which assumes that an objective reality exists, and considers that a faithful representation of the world ought to be achieved (Hadar and Soffer, 2006). Taking this view, hard modeling helps to achieve a faithful model of a WebApp.

In this category, we find two subcategories. One includes generic notations such as UML (Booch *et al.*, 1996), IDEF (Kim *et al.*, 2003) and TOSCA (Tomlinson *et al.*, 2012). In particular, UML is often used to design in detail specific aspects of WebApps, such as data and navigational structures (Rossi *et al.*, 2007). A major advantage of generic notations is that they are part of the cultural and educational background of most developers (The Joint ACM/AIS IS2020 Task Force, 2021). They can also be useful when development is outsourced (Krancher, 2020). However, BizDevOps teams may not have good incentives to use these notations. One reason is that modern WebApps use highly complex, dynamic components, which can be difficult, if not impossible, to model faithfully. In fact, a perceived need to model all potentially legal interactions of loosely coupled and dynamic components can act as a constraint, rather than an enabler, of agility and flexibility.

For simplicity of argument, just consider the front-end component of a WebApp, which is the component that users interact with. Because front-ends are expected to be highly interactive, flexible, and personalized, they are challenging to model (Bozzon *et al.*, 2006). If we consider the back-end component of a WebApp, which is the invisible component that provides logic, functionality and resources, the situation is similar. Back-ends are expected to integrate a myriad of loosely coupled pieces (e.g., microservices), which can be difficult if not impossible to model (Valderas *et al.*, 2020). Finally, the continuous nature of BizDevOps also makes it difficult to justify the nuisances of hard modeling. In particular, when time is critical, models lose in comparison to more expedite approaches like sketches (Yu *et al.*, 2018).

The other subcategory concerns model-driven engineering (MDE), where software is generated using standardized models and procedures (Brambilla *et al.*, 2006). An advantage of MDE is that the substantial modeling effort can be compensated down the line through automatic code generation (Aragón *et al.*, 2012; Siau *et al.*, 2022). Several authors have reviewed and compared existing MDE approaches (Aragón *et al.*, 2012; Molina-Ríos and Pedreira-Souto, 2020; Wakil and Jawawi, 2018). Molina-Rios and Pedreira-Souto (2020) note that, even though MDE is highly cited in the research literature, it seems almost not utilized by developers. The reasons for this lack of traction are varied and include lack of flexibility, pragmatism, use of hybrid technologies, and technical oddities (Bergmayr *et al.*, 2018; Molina-Ríos and Pedreira-Souto, 2020). There is a place for MDE approaches. However, MDE seems limited to niche purviews where developers need to define precise models for applications used by multiple stakeholders, and when software development has to be precisely documented and audited (Whittle *et al.*, 2014). For example, a major public health organization in the hometown of one of the authors is experimenting with MDE. Even so, where MDE has been successful, it has sometimes been criticized for excessive complexity, promoting basic functionality, and lacking flexibility and dynamism (Antunes and Tate, 2022b; Grolinger *et al.*, 2014; Poelmans *et al.*, 2013).

In summary, hard modeling approaches promote extensive analysis, formalization and standardization, and have limited application outside specific environments. In most cases, they conflict with the BizDevOps agile and flexible practices.

Lightweight modeling approaches

Lightweight modeling approaches adopt an epistemologically relativist position, which assumes that the world cannot be represented in objective terms, and therefore what counts is the interpretation of reality (Hadar and Soffer, 2006). Taking this view, faithful representation is eschewed to focus more on the cognitive and intuitive experiences of model users (Alter and Bork, 2019).

A paradigmatic example of lightweight modeling is given by the increasingly popular canvas models, which have been applied to a variety of domains (Antunes and Tate, 2022a). A canvas model represents a domain using a simplified ontology with few key concepts. Users make strategic decisions by integrating the canvas model with their tacit knowledge about the domain. Value modeling (mapping) is another example. Its main purpose is to analyze system requirements by tracking where value is coming from (Gordijn and Akkermans, 2003; Kartseva *et al.*, 2005). It is a lightweight approach because it does not try to faithfully represent value generation. Instead, it identifies loosely coupled concepts, which encapsulate knowledge about the system and the domain.

The trend towards lightweight modeling has been increasing in software development (Siau *et al.*, 2022). For instance, personas, agile user stories, storyboards, scenarios, empathy maps, stakeholder maps, affinity maps, and journey maps are often adopted to guide software development (Wood *et al.*, 2021). They help developers by focusing on key concepts like opportunities, pain/gain points and value.

Prior research on models adopting a canvas style provides a framework of understanding of the underpinnings of lightweight approaches (Antunes and Tate, 2022a). Firstly, a lightweight approach operates at a strategic level, where users work with a small set of key concepts. Secondly, lightweight approaches rely on tacit knowledge. Thirdly, they take a general view over the system. Finally, the general view integrates static and dynamic aspects, the former identifying key system concepts, and the latter helping to understand strategic changes. Later, we use this foundation to define a set of requirements for our approach.

Related research

As far as we know, no other lightweight approaches address WebApp design by BizDevOps teams. Nevertheless, we find in the literature some related propositions. Wang et al. (2021) proposed a lightweight approach to model microservices, a popular WebApp architecture. The approach uses three different and complex models, which make it difficult to use by non-technical users. TOSCAdata is a lightweight approach to model cloud applications, which is also exclusively focused on technical issues. Wakil et al. (2018) extend an existing hard modeling approach with elements and concepts that help developing WebApps; however, the hard nature of the approach does not make it amenable for BizDevOps. Papazoglou and van den Heuvel (2011) discuss how to blueprint modern WebApps; however, the focus is exclusively on back-end architectural issues. Integration between Biz, Dev and Ops is not considered. Karagiannis et al. (2022) discuss an integrated modeling ecosystem for digital innovation. The proposition consists of a laboratory environment for idea development, exploration and prototyping, it does not target production environments. Abiteboul et al. (2008) make an interesting proposition to model WebApp mashups, i.e., compositions that pull up resources and functionality from various sources. However, the model is technical and formal, therefore not amenable for BizDevOps.

Overall, prior studies do not address the requirements of lightweight modeling for BizDevOps. Next, the characteristics that such an approach should possess are defined.

Requirements

To guide the proposed approach, four requirements are defined. It should 1) offer a general plan; 2) be componentized; 3) define abstract features; and 4) cover a wide range of BizDevOps activities.

General plan

The approach should be capable of operating at a strategic system or application level, delivering a general plan, which BizDevOps teams can use for strategizing, prioritizing and decision-making. Such a general plan is not intended to impact team productivity but instead support communication and cohesiveness. An advantage of operating at a strategic level is that it avoids conflicting with a variety

of tactical tools already used by BizDevOps to improve performance in specific areas (e.g., infrastructure automation, testing, monitoring, and recovery) (Senapathi *et al.*, 2018).

Componentization

WebApps are inherently complex. This complexity has many origins, including architectural complexity, code complexity, feature complexity, and method complexity (Auer *et al.*, 2021; Molina-Ríos and Pedreira-Souto, 2020; Wakil and Jawawi, 2018). Complexity brings significant implications to the approach. In particular, complexity cannot be completely hidden, otherwise the approach will fail for excessive simplicity and lack of sufficient adherence to reality.

A way to handle complexity is to work with components. Componentization helps building a coherent frame of understanding using different levels of detail, which are articulated through abstraction and decomposition. A departing point to address the complexity of WebApps is to separate them in two components, front-end and back-end, where the front-end deals with the user interactions and the back-end deals with loosely-coupled service offerings (Shropshire *et al.*, 2018). This particular componentization is common in WebApp research and practice (Bonura *et al.*, 2002; Northwood, 2018). A componentized approach does not impact actual WebApp development, as it operates at the conceptual level. Instead, it helps teams managing the different layers of complexity of WebApps.

Abstract features

The WebApp landscape is typically unsteady. Unlike knowledge areas where the main discourse is structured by the research community, WebApps are mainly influenced by a continuous flux of technologies coming from practitioners, open-source initiatives and software vendors. This unsteadiness has significant implications on the modeling approach. The approach is expected to offer a frame of reference for WebApp development that is accepting of new and evolving trends. This can be accomplished using abstract features, where the focus is not on a particular technology, but instead on the essential elements behind them. For instance, rendering (presenting application contents on the screen) is an abstract feature, which can be implemented in different ways depending on the technology

(e.g., server-side and client-side rendering) (Duldulao and Cabagnot, 2021). The particularities of each implementation should not affect modeling.

Wide-ranging

An essential characteristic of BizDevOps is that it brings together activities that used to be siloed, with different teams dealing with business requirements, design, development, and operations (Ebert *et al.*, 2016). As these activities are brought together, a wide-ranging method is required to cover the whole set of practices collaboratively done by the team. As traditional boundaries between siloed activities blur, the approach should be expected to be evaluated less in regard to any specific activity, and more in regard to its efficacy in supporting shared responsibility for WebApp development.

The four requirements defined above play two important roles in this study. First, they provide a solution framework, which guides the method design. Second, they provide a baseline for evaluating the study. Next, the proposed blueprint is detailed.

BizDevOps Blueprint

The explanation of the blueprint is divided in two parts. First, a set of model components is defined, which provide the foundation for conceptualizing a WebApp. Second, the defined components are connected to define a strategic, high-level view of the WebApp.

The discussion is supported by a running example, which considers a stock trading application named eTrader. Briefly put, eTrader allows users to buy/sell stocks on the stock markets and perform related actions, such as following market trends and managing a portfolio.

Blueprint components

The two main components of the blueprint are the front-end and back-end (Figure 1). This separation does not imply that WebApps have to be strictly divided this way. In fact, this is just a logical separation, which is characteristic of componentization.

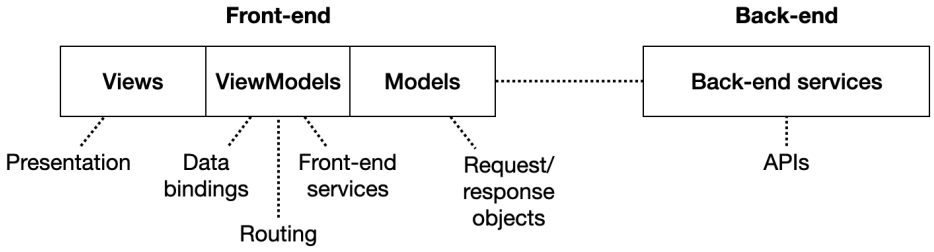


Figure 1: Blueprint components

The back-end adopts the backend-as-a-service abstract feature to provide a collection of independent services to applications (Dudjak and Martinović, 2020). Each service gives access to a set of assets through an API (De, 2017; Dudjak and Martinović, 2020). This feature is adopted because it is flexible, in particular in relation to the granularity of services. Decreasing granularity leads to microservices, which have been adopted by many modern WebApps (Larrucea *et al.*, 2018). Considering eTrader, it requires several back-end services: *market* provides information about the stock markets; *trade* allows users to buy and sell stocks; and *account management* handles user authentication and preferences.

Only one essential aspect of back-end services appears in the blueprint: the APIs. An API establishes a contract, which defines how a service is requested and responded, and the data exchanged (Dudjak and Martinović, 2020). Many developers take an API-first approach to WebApp development. A recent report indicates that 10% of developers' time is spent on API design (Postman, 2021). API design tools such as Swagger are popular and make it easy to design back-end services. Such designs can be easily integrated in the blueprint using tables (discussed later).

The front-end captures the user-facing functionality of a WebApp. This includes, in general terms, rendering data to be displayed, supporting user interactions, implementing the presentation logic, and handling events (Bonura *et al.*, 2002). In the case of eTrader, the front-end handles all user interactions required to log in the system, explore the stock markets, get details about specific stocks, buy and sell stocks, etc. It also handles events such as market updates and order updates.

The front-end adopts the Model-View-ViewModel (MVVM) abstract feature (Garofalo, 2011). MVVM was selected because it offers a flexible way to characterize modern front-ends, which does not depend on particular technologies (Bragge, 2013; Garofalo, 2011). According to MVVM, the front-end is

decomposed into three types of components: Models, Views and ViewModels. Models handle the integration with the back-end services. Views handle the user's interactions at a low-level of detail, which includes presentation, managing data inputs/outputs, and low-level processing (e.g., confirmations). ViewModels define the presentation logic at an abstract level of detail, working with Models and abstractions of Views to define how the WebApp behaves.

MVVM does not constrain the selection of specific WebApp technologies, given that specific components can be mapped to any abstract MVVM component. For instance, the template classes defined by Angular and React can be mapped to Views, as they define what is rendered on the screen (He, 2022).

What exactly should be modeled in relation to Model, View and ViewModel has to be defined. Considering Views, the blueprint only considers static presentations (user interface elements and layout). Even though Views usually include dynamic aspects, they are not considered in the blueprint, because they are too detailed to be useful in a general plan of a WebApp. Models provide domain representations, which are exchanged between the front-end and back-end through request/response objects. For instance, eTrader needs to exchange stock quotes and buy/sell orders between the front-end and back-end. Even though Models comprise both data and data processing functionality, the functionality of Models is not considered in the blueprint, as it is too specific (e.g., data conversions).

ViewModels define three abstract features, which can be modeled separately. One feature is binding abstract data to Views. By operating on the View through abstract data, the ViewModel reduces coupling with the View (Garofalo, 2011). It also reduces coupling with the Model. For instance, in eTrader, the developers may design a ViewModel that gets full details about a particular stock from the Model, but only delivers a selection to the View, which may depend on the interaction context.

Another feature is routing, which defines how users interact with the application by going through different presentations. For instance, in eTrader, users should be able to route between exploring the stock markets and placing buy/sell orders. However, because modern WebApps require interaction flexibility, routing can be overcomplicated. In some cases, routing is so complex that it requires

dedicated components to manage it (e.g., Redux and NgRx). More specific details on how the blueprint addresses these issues will be discussed in the next section.

The third and final feature concerns the presentation logic, which defines what happens when an event occurs. The front-end-as-a-service abstract feature is adopted to model presentation logic. Each front-end service defines which actions should occur after the service is invoked. For instance, eTrader requires front-end services to overview the stock market, update the details about a particular stock, put a buy/sell order, overview the portfolio, manage account, etc.

Regarding the front-end as a collection of services is an abstraction, which accommodates a variety of ways in which the presentation logic can be implemented. For instance, the presentation logic can be executed locally (e.g., browser) or remotely (e.g., web server and cloud). Micro-frontends are another approach, which takes ideas from back-end microservices. It divides the front-end into independent components (e.g., searching, exploring and trading in eTrader) (Pavlenko *et al.*, 2020). Each micro-frontend can be regarded as a combination of Model, View and ViewModel.

Connecting the blueprint components

The blueprint components discussed above have to be connected in some way. One starting point is the front-end. This starting point is not necessary or even suggested for every development; depending on specific contexts, BizDevOps teams may decide to start elsewhere.

To connect the front-end components, a set of principles defined by Jacobson et al. (2016) is followed: keep it simple by telling *stories*; understand the big picture; and focus on value. A story captures the essential aspects of how users achieve certain goals. eTrader can be described using three stories: authentication (login and sign-in); trading (buy and sell stocks); and account management (including money transfers). Each story has to be outlined. The focus is not on the fine-grained details, but giving a strategic view over what users do with the WebApp. For that reason, a high-level navigational map, or story space, is adopted to define stories (Schewe and Thalheim, 2019). The story space defines a set of abstract *places* and shows how users go from one place to another to accomplish goals. In each place, users can perform a set of *actions*.

According to this view, a WebApp is just a collection of places and potential actions connected by stories. Considering eTrader, the Trading story requires three places: Market, where users check what is going on with the stock markets; Trade, where users buy and sell stocks; and Portfolio, where users manage the portfolio of stocks. Figure 2 outlines the trading story (oval shape) as a collection of connected (arrows) places (rectangular shape).

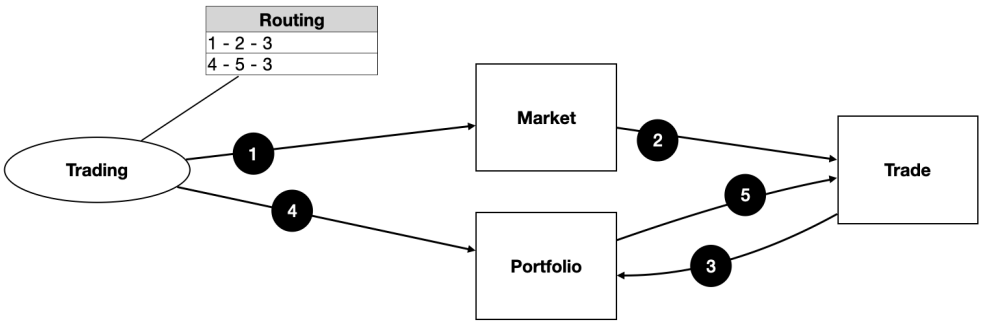
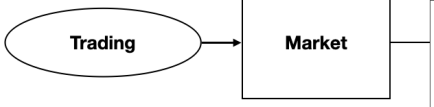


Figure 2: Story space for the trading story in eTrader

Using a WebApp can be seen as going through a set of stories. Considering eTrader, users can take two different paths: go to Market, select a particular share, and Trade that share; or instead, go to Portfolio, pick a particular share, and Trade that share. In Figure 2, these options are modeled using a routing table and index numbers attached to arrows between places. Given the lightweight nature of the approach, many routing details are not modeled explicitly, as they are not necessary. For instance, it is common that users can go back one step, or jump to the beginning of a story. Users also have to move somewhere when a story finishes. These capabilities can be assumed without compromising the usefulness of the blueprint. These details are intentionally absent to keep the blueprint as simple as possible.

Having defined some essential aspects of the front-end (stories, places and routing), it is now linked to the back-end. This is accomplished through places, which provide a logical structure focused on the users' goals. For each place, there is a set of back-end services, which are described using a table. The table defines: service name, type of call, endpoint, request, response and description. The type of call adopts RESTful, which is a popular approach to access back-end services (Pautasso, 2014). RESTful defines a set of calls including GET, PUT, DELETE, and POST. Figure 3 shows the back-end services table for the Market place in the Trading story of eTrader. The table defines the Market service, which has calls for selecting a particular stock exchange, searching, getting a list of stocks, getting details about

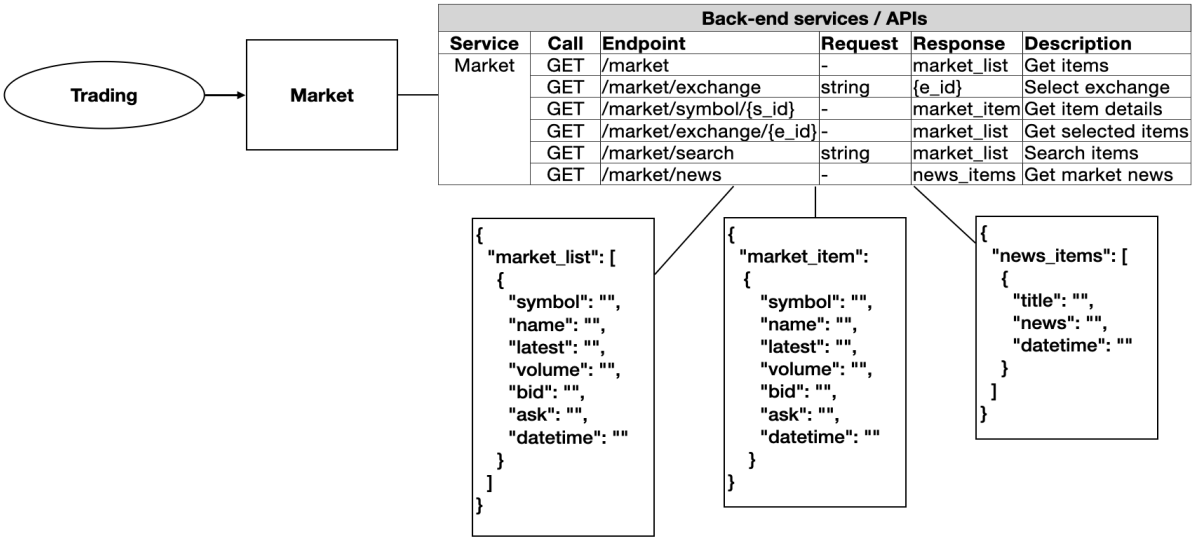
a particular stock, and getting the market news. These calls allow users to do several actions while in the Market place.



Back-end services / APIs					
Service	**Call**	**Endpoint**	**Request**	**Response**	**Description**
Market	GET	/market	-	market_list	Get items
	GET	/market/exchange	string	{e_id}	Select exchange
	GET	/market/symbol/{s_id}	-	market_item	Get item details
	GET	/market/exchange/{e_id}	-	market_list	Get selected items
	GET	/market/search	string	market_list	Search items
	GET	/market/news	-	news_items	Get market news

Figure 3: Back-end services table for eTrader's Market place

The back-end services table identifies a set of endpoints (unique identifiers) of an API, along with the required request/response objects. The endpoints follow the popular OpenAPI specification. The table can be substituted by schemes generated by API design tools like Swagger. The request/response objects can be further detailed. Figure 4 shows the request/response objects required by eTrader's Market place back-end services: market_list has a list of stocks; market_item has details about a stock; and news_items has a list of market news titles.



Back-end services / APIs					
Service	**Call**	**Endpoint**	**Request**	**Response**	**Description**
Market	GET	/market	-	market_list	Get items
	GET	/market/exchange	string	{e_id}	Select exchange
	GET	/market/symbol/{s_id}	-	market_item	Get item details
	GET	/market/exchange/{e_id}	-	market_list	Get selected items
	GET	/market/search	string	market_list	Search items
	GET	/market/news	-	news_items	Get market news

Figure 4: Request/response objects for eTrader's Market place

Having deviated from the front-end to the back-end, we now go back and complete the front-end. First, the MVVM elements have to be tied together. As noted earlier, Models link the front-end to the back-end. Therefore, they correspond to the request/response objects discussed above. Views materialize places, providing presentations for users to interact with. ViewModels provide routing between places, and let users perform certain actions at each place.

Instead of modeling presentations in an abstract way, the blueprint assumes that presentations can be generated by prototyping tools. The blueprint only links these presentations to places. Figure 5 shows presentations for the Market and Trade places in eTrader. These presentations were developed using Nicepage.

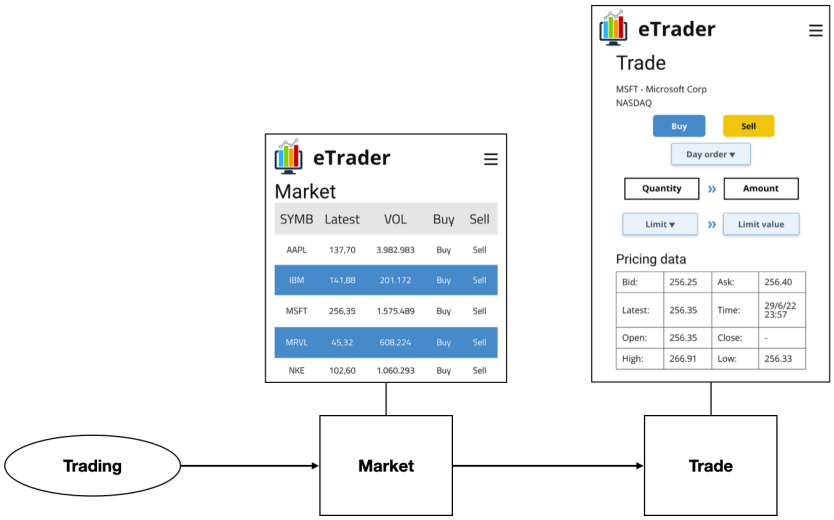


Figure 5: Presentations for eTrader's Market and Trade places

Only one primary presentation is linked to a place. Once again, this reflects the lightweight nature of the approach. Modern WebApps often have a myriad of secondary, often transient presentations, including pop-ups, notifications and confirmations. Trying to include all presentations in a single blueprint seems impractical. The adopted tradeoff is to emphasize relevance over detail, focusing on primary presentations.

Following MVVM, each presentation should be abstracted by the associated ViewModel through a data binding object. Therefore, data binding objects are linked to presentations. Figure 6 shows presentations and data binding objects for the Market and Trade places in eTrader.

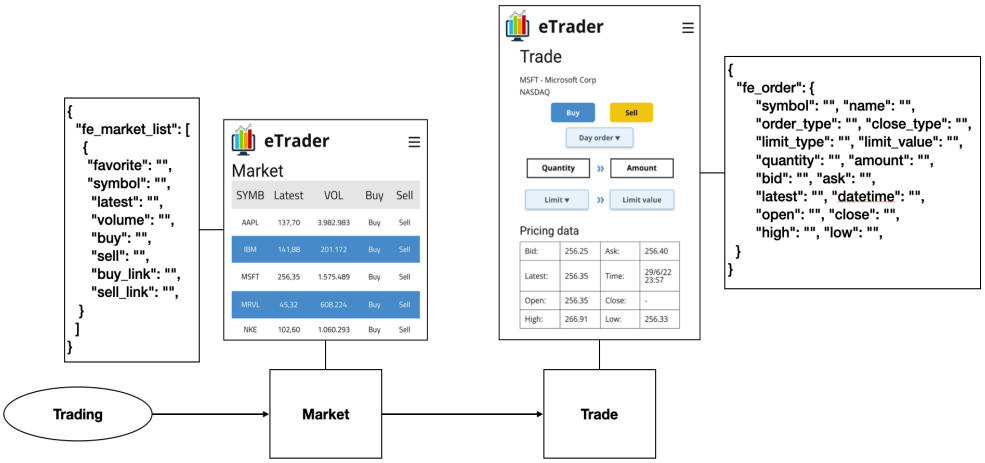


Figure 6: Presentations and data binding objects for the Market and Trade places in eTrader

As already noted, users interact with the front-end in two different ways. One involves routing and has already been discussed: routing tables show how users go to places. The other way concerns the local actions that users can execute in each place. These local actions are defined in a front-end services table. Figure 7 shows the front-end services table for the Portfolio place in eTrader, which includes: get portfolio, which displays the list of stocks held by the user; get summary, which provides the total value of stocks and profit loss (P/L) statement; and buy/sell, which routes the user to the Trade place.

Figure 7: Front-end services for eTrader's Portfolio place

In summary, the blueprint is a map that overviews the front-end and back-end using the MVVM abstraction. Stories and places play central roles in the blueprint, as they provide a strategic view over the WebApp functionality. Even though the blueprint has a variety of components, it is scalable because it can be simplified by omitting specific components (e.g., request/response and data binding objects). It can also be split in different stories. Different tradeoffs between business and technical views can also be achieved by giving more or less emphasis to the front-end or the back-end.

In Figure 8, we show eTrader's blueprint for the trading story. We note again that the blueprint does not have to be constructed in a definite way, following specific steps. Its main purpose is to bring the BizDevOps team on the same page, sharing a strategic view over the WebApp, integrating business and technical viewpoints.

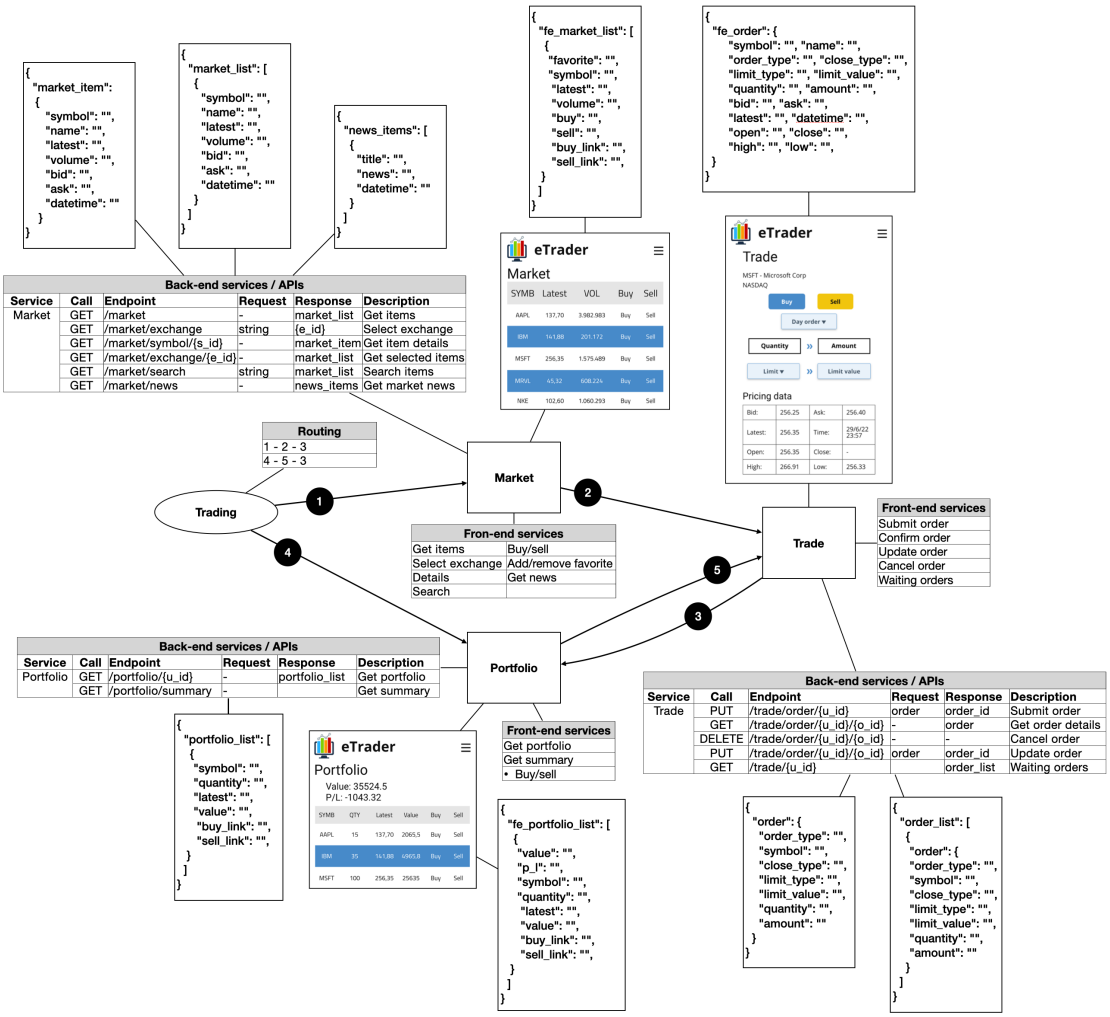


Figure 8: Blueprint of eTrader's trading story

Evaluation

Venable et al. (2016) suggest that with innovative artifacts like the blueprint both formative and summative evaluations can be applied. Following this suggestion, the evaluation of the blueprint involved formative and summative stages, which are discussed below.

Formative evaluation

The blueprint was developed along with teaching a course on WebApp development in a masters' in informatics. The learning goals involved advanced knowledge of modern web architectures, WebApp frameworks, cloud computing, and related concepts such as BizDevOps. The course requires teams to design a WebApp and submit a blueprint to explain the design. Around 60 students enroll in the course every year. Students have very diverse backgrounds, including social sciences, natural sciences,

business, education, IT, and software engineering (the majority). The approach was developed and matured in two course iterations. Discussions with students allowed to identify major weak points, and improve the structure, presentation and overall coherence. In particular, the blueprint was adjusted to support different design strategies, as some teams started with API design, and others started with presentation. The wide variety of students' backgrounds also allowed to balance the business and technical views offered by the blueprint.

Summative evaluation: Interviews with experts

The summative evaluation concerned a set of interviews with experts working in (Biz)DevOps environments, with the purpose to gather evaluative feedback about the blueprint. The choice of the interview method is appropriate as it enables experts to discuss their thinking and provides in-deep discussions about the blueprint, and it could be utilized by them. Further, interviews are considered suitable for evaluating new models, as suggested by Syed et al. (2019) regarding the evaluation of the identity management value model.

Key experts working in different roles across a variety of (Biz)DevOps projects were recruited for the interviews. Considered roles included back-end, front-end and full-stack development, and project management. Two criteria were used to invite participants: 1) minimum of 1 year experience on the job; and 2) working in a (Biz)DevOps environment.

Semi-structured interviews were used for data collection. The process was organized in the following way. Before asking any specific questions, the blueprint was introduced to the participant. Then, a set of questions were raised: "Do you think the blueprint would be useful in your projects? How?", "Would the blueprint accommodate the project's specific architecture and technology?", "What do you think about the link between front-end and back-end in the blueprint?", "Do you think the blueprint could give a complete overview of your project?", "Is there anything that the blueprint does not address?", and "Would you adopt the blueprint? Why?". Given the relatively straightforward nature of these questions, we reached saturation after five interviews, as the same feedback started to be repeated by the participants (Saunders *et al.*, 2018).

All interviews were conducted online through MS Teams, using audio recording, and they took 30-45 minutes. The audio recordings were analyzed by the researchers using content analysis tools. Table 1 summarizes the participants' profiles.

Table 1: Participant's profiles

ID	Roles	Experience	Gender
A	Back-end developer, tester	>3 years	Male
B	CIO, project manager, full-stack developer	>12 years	Male
C	Operational officer, developer	>2 years	Male
D	Technical lead (both front-end and back-end)	>2 years	Male
E	Front-end developer	>4 years	Male

A cross-interview procedure was adopted for data analysis (Patton, 2014), focusing on three aspects: evaluative elements (e.g., usefulness); feedback about specific blueprint components (e.g., ViewModels); and issues/problems (e.g., complexity).

As shown in Table 2, the participants provided a set of comments and concrete remarks that indicate the model is useful, understandable, and it provides an overview picture. Indeed, most participants asserted evidence to support these evaluative elements. Most participants would also consider adopting the model in the future.

Table 2: Interview results

Evaluative elements	Selected feedback
Usefulness	+ "The good thing about this model is that when there are changes in one of the front-end services, these changes are not moved through all the application. It just has to focus on that particular service" (A) + "The model maps with my working experience. It is comprehensive" (B) + "I think the model is suitable regarding the discussion [communication] between different teams" (C) + "It is similar [to what we are doing]" (E)
Strategic view	+ "all of these are independent, […] but there's always a part in which all of them have to be linked" (A) + "The model provides an overview picture, yet may not represent the details" (B) + "The model stays in an abstract level" (C) + "The model is enough" (D)
Understandability	+ "It is ok. I think it is enough" (A) + "The model is simple and understandable" (B) = "If we use the model, does it take more time for small projects?" (B) + "I think it is enough to design a [web] application" (C) = "As I am already familiar with MVC, I can understand the model instantly" (E)
Future adoption	+ "Yes, one of the most advantageous things that this model has is that it can be recycled, especially in the back-end" (A) + "The model can also be applied for mobile application development" (B) + "I will use this model as reference to discuss with other teams" (C) − "It is hard to say, as the model needs more components such as the communication between front-end and back-end" (D)

	+ "I want to try it" (E)
Model components	**Selected feedback**
Views: Presentation	+ "[When we start,] we design the layout, user interface, and mock-up prototypes" (B) − "If we apply this model for mobile applications, View needs to be adapted according to mobile types" (B) + "It is certain that we have dynamic elements, yet we also need to identify static elements" (C) = "Front-end [business analytics and designers] will develop the mock-up" (D) + "Sometimes, we use Figma for the interface presentation" (E)
ViewModel: Data bindings	− "I think it should show the databases" (C) + "In my project, front-end developers communicate with back-end to configure data bindings" (D)
ViewModel: Routing	− "Routing may be hard to understand" (B) − "There may also be routing between different versions of deployments" (C) = "Front-end designers define routing paths" (D)
ViewModel: Front-end services	+ "Developers can write different [front-end] services in the controller [ViewModel]" (B) = "We mainly do microservices" (D)
Model: Req/resp objects	+ "As soon as we make the front-end that communicates with the APIs, it turns out to be open" (A)
Back-end: APIs	+ "We define the APIs" (B) + "We link the back-end and front-end through contracts and APIs (C) = "Back-end links with front-end through APIs" (E)
Back-end: Services	+ We can take it as individual services (A) = "We need to identify in the back-end, whether it goes with monolithic or microservices" (C)
Issues/ problems	**Selected feedback**
Technical complexity/ simplicity	− "It is too much detail" (A) + "The model is simple and understandable" (B) − "[The model] may be more about development than operations, as there is no operational component here" (C) − "The model is too simple. More details are needed" (D) + "The model is detailed enough" (E)
Front-end abstraction	− "Change the model into MVC" (B) = "Some WebApps use MVC or MVVM" (D)
Front-end, back-end link	− "There should be another service in the middle for authentication" (A) − "There are more than one back-end and there are more than one front-end" (A) − "It lacks parts for deployment" (D) − "Font-end calls back-end through a gateway, which may also take into account load balancing" (D)
Presentation	= "There should be UI/UX elements" (C)
Continuous delivery	= "Continuous integration and continuous delivery is important in DevOps projects" (D)

Note: '+' indicates supporting evidence; '-' negative evidence; and '=' suggests there are some concerns.

Regarding the model components, the participants highlighted a clear representation of the model components. On the other hand, the participants provided a range of viewpoints. Of these, routing seems to be the most polarizing aspect of the model. We note that due to the participants' different roles in their projects, they may be more familiar with certain components than others.

Regarding problems/issues, the participants identified several concerns with the proposed model, of which two stand out most. First, the participants have opposing views about the model's technical complexity: several suggested that the model is too detailed, while others suggested it is simple enough. This issue may come from the different projects discussed in the interviews, which have varied levels

of complexity. Second, some participants suggested changing the front-end abstraction to other abstractions, which they are familiar with. While noting the suggestions, we kept the MVVM abstraction in the blueprint, as it is flexible and widely recognized by WebApp developers (Li *et al.*, 2022).

Overall, the experts asserted that the blueprint is useful, understandable and ready for adoption in future projects.

Discussion

Four requirements have been outlined to address the research problem. It is now time to check if the blueprint satisfies the requirements. Furthermore, the blueprint is also contrasted against hard modeling and other lightweight modeling approaches mentioned in the background section (Table 3).

Table 3: Contrasting the blueprint alongside other approaches

Requirements	**Blueprint**	**Hard modeling approaches**	**Other lightweight modeling approaches**
General plan	Covers essential aspects of WebApps at a strategic level of detail. Reduces technical complexity, while retaining essential technical aspects. The strategic focus aligns well with BizDevOps.	Fragmented in a variety of models targeting specific issues (e.g., use cases, classes, activities, data pipelines). The various models can be difficult to connect. Models emphasize technical complexity, detail and formalisms, rather than strategic view. For these reasons, hard modeling does not align well with BizDevOps.	Tend to focus on specific concerns, rather than providing a general plan. Bridges between business and technical perspectives may be unclear. BizDevOps teams may use these approaches in specific stages of development, but not in a continuous way.
Componentization	Defines a small number of key WebApp components, which are clearly connected. Changes can be tracked across the whole WebApp. These characteristics make it easier for BizDevOps to strategize.	Componentization is promoted, but at a technical, low level of detail, which makes it difficult to discuss from a business perspective. Changes are problematic to address, as they may impact various models. Not flexible enough in agile environments like BizDevOps.	Tend to be centered on upfront idea development and design, and therefore lack the technical aspect. Integration between Biz, Dev and Ops is not considered.
Abstract features	Adopts key features of modern WebApps in a simple way. Simplicity and clarity make it easier for BizDevOps to design WebApps.	Generic notations lack a clear, simple frame of reference for WebApp development. MDE approaches solve this problem, but lack flexibility.	Conceived to support the design process, regarding WebApp development in a generic, abstract way. As such, they relate more to Biz than Dev and Ops.
Wide-ranging	Covers the whole WebApp development, from Biz to Dev and Ops.	Mainly focused on the technical viewpoint. Even more business-oriented models can become too technical (e.g., use cases). For that reason, it does not align well with BizDevOps.	Often considered useful when doing upfront activities (e.g., idea exploration). They lack a continuous link to the BizDevOps pipeline.

The first requirement is that the blueprint should provide a general plan. This is accomplished by the blueprint by aligning business elements like stories and places, with technical elements like services,

request/response objects and APIs. The blueprint shows technical details, however in a way that does not overshadow the general plan, where the WebApp is envisaged as a collection of stories and places.

The blueprint contrasts with hard modeling approaches, which require a variety of models with different perspectives and levels of specificity to understand a system (Alter and Bork, 2019). This fragmentation makes it difficult to connect elements, and in turn makes it difficult to design (and re-design) WebApps at the strategic level. Bucchiarone et al. (2021) note that the sheer complexity of modern systems makes it difficult to deal with heterogeneous collections of models. Against this backdrop, the blueprint provides a simple and consistent approach to manage WebApp complexity.

A variety of lightweight modeling approaches have been adopted by developers, e.g., storyboards (Čok *et al.*, 2022), value stream mapping (Matt, 2014), and journey mapping (Marquez *et al.*, 2015). However, they reduce complexity by focusing on specific stages (e.g., design-time) and goals (e.g., requirements definition) (Karagiannis *et al.*, 2022), which make them difficult to use by BizDevOps throughout the whole development process.

The second requirement is that the blueprint should handle complexity by working with components. This is accomplished by defining a small set of key components required by WebApps, which are divided between front-end and back-end, with the front-end further divided into Models, Views and ViewModels. These components are clearly connected by the blueprint, which allows BizDevOps teams to check interdependencies and track changes across the whole component set. This contrasts with hard modeling approaches, where checking interdependencies, tracking changes, and making comparisons can be challenging, as multiple models with high technical complexity are involved (Bucchiarone, Cabot, *et al.*, 2020; Karagiannis *et al.*, 2022). Hard modeling is also regarded as not flexible enough in agile environments (Badreddin *et al.*, 2018), such as BizDevOps. This happens because hard models require significant effort upfront. The blueprint requires much less effort. As for other lightweight approaches, they tend to be centered on design thinking and upfront development of ideas (Badreddin *et al.*, 2018) and therefore lack integration with Dev and Ops.

The third requirement is that the blueprint should focus on abstract features. This is accomplished by adopting key features of modern WebApps, including: front-end and back-end split, back-end services

(Bucchiarone, Dragoni, *et al.*, 2020), API design (Dudjak and Martinović, 2020), front-end services (Pavlenko *et al.*, 2020), and MVVM (Sorensen and Mikailesc, 2010). The integration of these abstract features is made in a simple way, using simple concepts like stories and places. Such simplicity helps BizDevOps teams to work on WebApps by putting together and aligning a set of components. In contrast, hard modeling generic notations have been conceived to be abstract and complete (Wand and Weber, 2002). They lack a clear, simple frame of reference for WebApp design. The required levels of specificity and rigor, along with the lack of focus on WebApps, make it difficult for different stakeholders, in particular non-technical people, to use them (Alter and Bork, 2019; Bucchiarone, Cabot, *et al.*, 2020). In fact, industry surveys indicate that even most technical stakeholders primarily use hard models to think and communicate informally, often preferring sketches to models (Bucchiarone *et al.*, 2021). The blueprint offers an alternative to both hard modeling and sketching. Considering other lightweight models, they tend to regard WebApp development in a generic, abstract way (e.g., lack of separation between front-end and back-end) and thus make it difficult to integrate business and technical communication. MDE approaches offer specific frames of reference for WebApp development. However, they bring their own challenges. In particular, they emphasize technical characteristics and are essentially focused on deployment technologies (Rademacher *et al.*, 2018).

The fourth and final requirement is that the blueprint should be wide ranging, covering Biz, Dev and Ops. Regarding Biz, the blueprint organizes WebApps around stories, which promote empathy, design thinking and value generation (Magare and Lamin, 2017; Siegel and Dray, 2019). Considering Dev, the blueprint organizes WebApps around spaces, services (front-end and back-end) and request/response objects. Finally, regarding Ops, the blueprint provides an agile way to track changes back and forth. In contrast, hard modeling approaches promote a technical viewpoint, focusing on architectural, structural and functional details (all separated), rather than an integrated business and technical viewpoint, which is aware of business concerns, addresses wicked managerial concepts (e.g., stories and places) (Roschnik and Missonier, 2022), and integrates technical concepts. Other lightweight approaches have been considered useful when doing upfront activities (e.g., design and idea exploration), but neglect the whole BizDevOps pipeline.

Conclusion

BizDevOps is an emergent concept, which is driven by the success of Agile and DevOps software development methodologies (DORA, 2021; Gearset, 2022). As an emergent concept, the methods, processes and tools supporting BizDevOps are still in flux. Nevertheless, they are already impacting the way WebApps are developed. In particular, increased development agility, flexibility and continuity, and increased integration between Biz, Dev and Ops, challenge traditional modeling approaches. This study identifies four requirements addressing these challenges. First, modeling should focus on the strategic level, where WebApps are designed and developed in close collaboration by different stakeholders. Second, WebApps have to be thought out using components that expose the systemic nature of WebApps, and in ways that are actionable by BizDevOps teams. Third, modeling has to embrace a diverse and volatile set of technologies required to develop WebApps, while still supporting close integration between Biz, Dev and Ops. To accomplish this, modeling has to focus on abstract rather than concrete features, around which BizDevOps teams can build consensus. Fourth and final, modeling has to be wide-ranging, covering the variety of activities expected of a BizDevOps team, some more business oriented (e.g., value propositions) and some more technically oriented (e.g., APIs).

The proposed blueprint tackles the abovementioned requirements in two steps. First, it defines a set of abstract components and combines them using a set of abstract features that, together characterize essential aspects of a WebApp at a strategic level. Second, it defines a set of flexible relationships that fuse the business and technical views (e.g., stories, places, actions, and services).

The blueprint does not impose precedence rules, development stages, or particular development processes. In fact, BizDevOps teams may adopt their own Biz, Dev and Ops strategies. They may also adopt specific technologies, as the blueprint is abstract enough to accommodate a variety of architectural choices, and front-end and back-end technologies. Nevertheless, there is a fine line between the business and technical views. If a model gets too technical, there is the risk of alienating the business stakeholders. If it gets too non-technical, there is also the risk of disaffecting the technical stakeholders. The blueprint offers a baseline, a way to define compromises, around this fine line. Teams can add,

modify or remove details without compromising the baseline. Furthermore, the notation adopted for the blueprint avoids intricate formalisms and is highly pragmatic.

In summary, the blueprint contributes a strategic map that offers a simple, flexible baseline for BizDevOps teams to design WebApps. Considering modeling epistemology, the blueprint is unique because it embraces a lightweight whole-of-a-system approach, which trades the faithfulness of the system representation in favor of the purpose of representation.

WebApps will continue evolving into even more complex, flexible, dynamic, and intertwined systems. Therefore, there is not much future for hard modeling in the two categories identified by this study, generic notations and MDE. Regarding the former, the main problem is the increasing tensions between representation and purpose in the modeling epistemology. Challenges in both representation and purpose are increasing, which cannot be easily addressed using generic notations. Regarding MDE, the approach is challenged by focus on low code (Bock and Frank, 2021) and xOps technologies, which adopt more agile and flexible paradigms. On the other hand, lightweight approaches offer the *simplexity* (Colville *et al.*, 2011) advantage: they adopt simple strategies to make sense of complex realities.

The summative evaluation indicates that the blueprint can be useful in providing an overview of WebApp projects. The evaluation also highlights a diversity of viewpoints. Nevertheless, instead of seeing this as an obstacle, we see it as an opportunity. That is, rather than standardizing approaches and practices, the blueprint encourages BizDevOps teams to share their different views in a flexible way.

Nevertheless, some limitations should be recognized. The blueprint is anchored in stories, as the main mediators between the business and technical views. However, increasing WebApp complexity can make it difficult to define useable stories. Future research is necessary to understand if there are advantages in anchoring the blueprint in even more flexible concepts, such as events (Antunes and Tate, 2022b). Future research is also necessary to study the blueprint in actual use by BizDevOps teams.

References

Abiteboul, S., Greenshpan, O. and Milo, T. (2008), “Modeling the mashup space”, *Proceedings of the 10th ACM Workshop on Web Information and Data Management*, pp. 87–94.

Alt, R., Auth, G. and Kögler, C. (2021), "DevOps for Continuous Innovation", *Continuous Innovation with DevOps*, Springer, pp. 17–36.

Alter, S. and Bork, D. (2019), "Work system modeling method with different levels of specificity and rigor for different stakeholder purposes", presented at the 14th International Conference on Wirtschaftsinformatik, Siegen, Germany.

Antunes, P. and Tate, M. (2022a), "Examining the Canvas as a Domain-Independent Artifact", *Information Systems and E-Business Management*, Vol. 20, pp. 495–514, doi: 10.1007/s10257-022-00556-5.

Antunes, P. and Tate, M. (2022b), "Business Process Conceptualizations and the Flexibility-Support Tradeoff", *Business Process Management Journal*, Vol. 28 No. 3, pp. 856–875, doi: 10.1108/BPMJ-10-2021-0677.

Aragón, G., Escalona, M., Lang, M. and Hilera, J. (2012), "An analysis of model-driven web engineering methodologies", *International Journal of Innovative Computing, Information and Control*, Citeseer, Vol. 8 No. 12, pp. 1–10.

Auer, F., Lenarduzzi, V., Felderer, M. and Taibi, D. (2021), "From monolithic systems to Microservices: An assessment framework", *Information and Software Technology*, Vol. 137, p. 106600.

Badreddin, O., Khandoker, R., Forward, A., Masmali, O. and Lethbridge, T. (2018), "A decade of software design and modeling: A survey to uncover trends of the practice", *Proceedings of the 21th Acm/Ieee International Conference on Model Driven Engineering Languages and Systems*, pp. 245–255.

Baresi, L. and Garriga, M. (2020), "Microservices: the evolution and extinction of web services?", *Microservices*, Springer, pp. 3–28.

Bergmayr, A., Breitenbücher, U., Ferry, N., Rossini, A., Solberg, A., Wimmer, M., Kappel, G., *et al.* (2018), "A systematic review of cloud modeling languages", *ACM Computing Surveys*, Vol. 51 No. 1, pp. 1–38.

Bock, A.C. and Frank, U. (2021), “Low-code platform”, *Business & Information Systems Engineering*, Vol. 63 No. 6, pp. 733–740.

Bonura, D., Culmone, R. and Merelli, E. (2002), “Patterns for web applications”, *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, ACM, pp. 739–746.

Booch, G., Rumbaugh, J. and Jackobson, I. (1996), *The Unified Modeling Language for Object-Oriented Development*, Rational Software Corporation, Santa Clara, California.

Bozzon, A., Comai, S., Fraternali, P. and Carughi, G. (2006), “Conceptual modeling and code generation for rich internet applications”, *Proceedings of the 6th International Conference on Web Engineering*, ACM, Palo Alto, California, pp. 353–360.

Bragge, M. (2013), *Model-View-Controller Architectural Pattern and Its Evolution in Graphical User Interface Frameworks*, LUT University, Finland.

Brambilla, M., Ceri, S., Fraternali, P. and Manolescu, I. (2006), “Process modeling in web applications”, *ACM Transactions on Software Engineering and Methodology*, Vol. 15 No. 4, pp. 360–409.

Bucchiarone, A., Cabot, J., Paige, R. and Pierantonio, A. (2020), “Grand challenges in model-driven engineering: an analysis of the state of the research”, *Software and Systems Modeling*, Vol. 19 No. 1, pp. 5–13.

Bucchiarone, A., Ciccozzi, F., Lambers, L., Pierantonio, A., Tichy, M., Tisi, M., Wortmann, A., *et al.* (2021), “What is the future of modeling?”, *IEEE Software*, Vol. 38 No. 2, pp. 119–127.

Bucchiarone, A., Dragoni, N., Dustdar, S., Lago, P., Mazzara, M., Rivera, V. and Sadovykh, A. (2020), *Microservices Science and Engineering*, Springer Nature, doi: 10.1007/978- 3- 030- 31646- 4.

Burton-Jones, A. and Grange, C. (2013), “From use to effective use: A representation theory perspective”, *Information Systems Research*, Vol. 24 No. 3, pp. 632–658.

Burton-Jones, A., Recker, J., Indulska, M., Green, P. and Weber, R. (2017), "Assessing representation theory with a framework for pursuing success and failure", *MIS Quarterly*, Vol. 41 No. 4, pp. 1307–1333.

Camilli, M., Bellettini, C., Capra, L. and Monga, M. (2017), "A formal framework for specifying and verifying microservices based process flows", *International Conference on Software Engineering and Formal Methods*, Springer, pp. 187–202.

Chasioti, K. (2019), *BizDevOps: A Process Model for the Alignment of DevOps with Business Goals*, Utrecht University.

Čok, V., Vlah, D. and Vukašinović, N. (2022), "Storyboards as an Engineering Tool for Extraction of Functional Requirements", *International Design Conference - Design 2022*, Cambridge University Press, pp. 2273–2282.

Colville, I., Brown, A. and Pye, A. (2011), "Simplexity: Sensemaking, organizing and storytelling for our time", *Human Relations*, Vol. 65 No. 1, pp. 5–15.

De, B. (2017), "API governance", *API Management*, Springer, pp. 179–188.

DORA. (2021), *Accelerate State of DevOps 2021*, DevOps Research and Assessment (DORA), Google Cloud.

Dudjak, M. and Martinović, G. (2020), "An API-first methodology for designing a microservice-based Backend as a Service platform", *Information Technology and Control*, Vol. 49 No. 2, pp. 206–223.

Duldulao, D. and Cabagnot, R. (2021), *Practical Enterprise React*, Apress Media, LLC.

Ebert, C., Gallardo, G., Hernantes, J. and Serrano, N. (2016), "DevOps", *IEEE Software*, Vol. 33 No. 3, pp. 94–100.

Erickson, J., Lyytinen, K. and Siau, K. (2005), "Agile modeling, agile software development, and extreme programming: the state of research", *Journal of Database Management*, IGI Global, Vol. 16 No. 4, pp. 88–100.

Garofalo, R. (2011), *Building Enterprise Applications with Windows Presentation Foundation and the Model View ViewModel Pattern*, Microsoft Press.

Gearset. (2022), *The State of Salesforce DevOps 2022*.

Giere, R. (2004), "How models are used to represent reality", *Philosophy of Science*, Vol. 71 No. 5, pp. 742–752.

Gordijn, J. and Akkermans, J. (2003), "Value-based requirements engineering: exploring innovative e-commerce ideas", *Requirements Engineering*, Vol. 8 No. 2, pp. 114–134.

Grolinger, K., Capretz, M., Cunha, A. and Tazi, S. (2014), "Integration of business process modeling and Web services: a survey", *Service Oriented Computing and Applications*, Vol. 8 No. 2, pp. 105–128.

Hadar, I. and Soffer, P. (2006), "Variations in conceptual modeling: classification and ontological analysis", *Journal of the Association for Information Systems*, Vol. 7 No. 8, p. 20.

He, M. (2022), *Creating Apps with React Native*, Apress Media, LLC.

Jacobson, I., Spence, I. and Kerr, B. (2016), "Use-case 2.0", *Communications of the ACM*, Vol. 59 No. 5, pp. 61–69.

Karagiannis, D., Buchmann, R. and Utz, W. (2022), "The OMiLAB Digital Innovation environment: Agile conceptual models to bridge business value with Digital and Physical Twins for Product-Service Systems development", *Computers in Industry*, Vol. 138, p. 103631.

Kartseva, V., Gordijn, J. and Tan, Y.-H. (2005), "Toward a modeling tool for designing control mechanisms for network organizations", *International Journal of Electronic Commerce*, Vol. 10 No. 2, pp. 58–84.

Kim, C.-H., Weston, R., Hodgson, A. and Lee, K.-H. (2003), "The complementary use of IDEF and UML modelling approaches", *Computers in Industry*, Vol. 50 No. 1, pp. 35–56.

Knuuttila, T. (2011), "Modelling and representing: An artefactual approach to model-based representation", *Studies in History and Philosophy of Science Part A*, Vol. 42 No. 2, pp. 262–271.

Krancher, O. (2020), "Agile software development practices and success in outsourced projects: The moderating role of requirements risk", *International Conference on Agile Software Development*, Springer, pp. 56–72.

Larrucea, X., Santamaria, I., Colomo-Palacios, R. and Ebert, C. (2018), "Microservices", *IEEE Software*, Vol. 35 No. 3, pp. 96–100.

Li, X., Wang, S., Liu, Z. and Wu, G. (2022), "Design and implementation of enterprise web application common framework based on model-view-viewmodel architecture", *Fifth International Conference on Mechatronics and Computer Technology Engineering*, Vol. 12500, SPIE, pp. 1074–1077.

Lohrasbinasab, I., Acharya, P. and Colomo-Palacios, R. (2020), "BizDevOps: A Multivocal Literature Review", *International Conference on Computational Science and Its Applications*, Springer, pp. 698–713.

Magare, A. and Lamin, M. (2017), "Cognitive evolution in software development life cycle through Design Thinking", *Computer Modelling & New Technologies*, Vol. 21 No. 3, pp. 31–34.

Marquez, J., Downey, A. and Clement, R. (2015), "Walking a mile in the user's shoes: Customer journey mapping as a method to understanding the user experience", *Internet Reference Services Quarterly*, Vol. 20 No. 3–4, pp. 135–150.

Matt, D. (2014), "Adaptation of the value stream mapping approach to the design of lean engineer-to-order production systems: A case study", *Journal of Manufacturing Technology Management*, Vol. 25 No. 3, pp. 334–350.

Molina-Ríos, J. and Pedreira-Souto, N. (2020), "Comparison of development methodologies in web applications", *Information and Software Technology*, Vol. 119, p. 106238.

Northwood, C. (2018), *The Full Stack Developer: Your Essential Guide to the Everyday Skills Expected of a Modern Full Stack Web Developer*, Springer.

Papazoglou, M. and van den Heuvel, W. (2011), "Blueprinting the cloud", *IEEE Internet Computing*, Vol. 15 No. 6, pp. 74–79.

Patton, M. (2014), *Qualitative Research & Evaluation Methods: Integrating Theory and Practice*, Sage publications.

Pautasso, C. (2014), "RESTful web services: principles, patterns, emerging technologies", *Web Services Foundations*, Springer, pp. 31–51.

Pavlenko, A., Askarbekuly, N., Megha, S. and Mazzara, M. (2020), "Micro-frontends: application of microservices to web front-ends.", *Journal of Internet Services and Information Security*, Vol. 10 No. 2, pp. 49–66.

Poelmans, S., Reijers, H. and Recker, J. (2013), "Investigating the success of operational business process management systems", *Information Technology and Management*, Vol. 14 No. 4, pp. 295–314.

Postman. (2021), *2021 State of the API Report*, Postman.

Rademacher, F., Sorgalla, J. and Sachweh, S. (2018), "Challenges of domain-driven microservice design: A model-driven perspective", *IEEE Software*, IEEE, Vol. 35 No. 3, pp. 36–43.

Recker, J., Indulska, M., Green, P., Burton-Jones, A. and Weber, R. (2019), "Information Systems as Representations: A Review of the Theory and Evidence", *Journal of the Association for Information Systems*, Vol. 20 No. 6, p. 5.

Roschnik, A. and Missonier, S. (2022), "A Framework Proposal to Evaluate Conceptual Models Framing Wicked Managerial Concepts", presented at the Thirtieth European Conference on Information Systems, Timișoara, Romania.

Rossi, G., Pastor, O., Schwabe, D. and Olsina, L. (2007), *Web Engineering: Modelling and Implementing Web Applications*, Springer Science & Business Media.

Samset, K. and Volden, G. (2016), "Front-end definition of projects: Ten paradoxes and some reflections regarding project management and project governance", *International Journal of Project Management*, Vol. 34 No. 2, pp. 297–313.

Sargent, R.G. (2015), "Types of Models", *Modeling and Simulation in the Systems Engineering Life Cycle*, Springer, pp. 51–55.

Sarker, S., Munson, C., Sarker, S. and Chakraborty, S. (2009), "Assessing the relative contribution of the facets of agility to distributed systems development success: an Analytic Hierarchy Process approach", *European Journal of Information Systems*, Vol. 18 No. 4, pp. 285–299.

Saunders, B., Sim, J., Kingstone, T., Baker, S., Waterfield, J., Bartlam, B., Burroughs, H., *et al.* (2018), "Saturation in qualitative research: exploring its conceptualization and operationalization", *Quality & Quantity*, Vol. 52, pp. 1893–1907.

Schewe, K. and Thalheim, B. (2019), *Design and Development of Web Information Systems*, Springer.

Senapathi, M., Buchan, J. and Osman, H. (2018), "DevOps capabilities, practices, and challenges: Insights from a case study", *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, pp. 57–67.

Shropshire, J., Landry, J. and Presley, S. (2018), "Towards a consensus definition of full-stack development", *Proceedings of the Southern Association for Information Systems Conference*, St. Augustine, IL, USA, pp. 1–6.

Siau, K., Woo, C., Story, V., Chiang, R., Chua, C. and Beard, J. (2022), "Information Systems Analysis and Design: Past Revolutions, Present Challenges, and Future Research Directions", *Communications of the Association for Information Systems*, Vol. 50 No. 1, p. 33.

Siegel, D. and Dray, S. (2019), "The map is not the territory: empathy in design", *Interactions*, Vol. 26 No. 2, pp. 82–85.

Sorensen, E. and Mikailesc, M. (2010), "Model-view-ViewModel (MVVM) design pattern using Windows Presentation Foundation (WPF) technology", *MegaByte Journal*, Vol. 9 No. 4, pp. 1–19.

Stack Overflow. (2022), *Stack Overflow Annual Developer Survey*.

Syed, R., Dhillon, G. and Merrick, J. (2019), "The identity management value model: A design science approach to assess value gaps on social media", *Decision Sciences*, Vol. 50 No. 3, pp. 498–536.

Tate, M., Bongiovanni, I., Kowalkiewicz, M. and Townson, P. (2018), "Managing the 'Fuzzy front end' of open digital service innovation in the public sector: A methodology", *International Journal of Information Management*, Vol. 39, pp. 186–198.

The Joint ACM/AIS IS2020 Task Force. (2021), *IS2020 A Competency Model for Undergraduate Programs in Information Systems*, ACM/AIS.

Tomlinson, B., Ross, J., Paul, A., Eric, B., Donald, P., Joseph, C., Martin, M., *et al.* (2012), "Massively distributed authorship of academic papers", *Proceedings of the 2012 ACM Annual Conference. Extended Abstracts on Human Factors in Computing Systems*.

Valderas, P., Torres, V. and Pelechano, V. (2020), "A microservice composition approach based on the choreography of BPMN fragments", *Information and Software Technology*, Elsevier, Vol. 127, p. 106370.

Venable, J., Pries-Heje, J. and Baskerville, R. (2016), "FEDS: a framework for evaluation in design science research", *European Journal of Information Systems*, Vol. 25 No. 1, pp. 77–89.

Wakil, K. and Jawawi, D. (2018), "A new adaptive model for web engineering methods to develop modern web applications", *Proceedings of the 2018 International Conference on Software Engineering and Information Management*, ACM, Casablanca, Morocco, pp. 32–39.

Wakil, K., Jawawi, D. and Rachmat, H. (2018), "Enhancing interaction flow modeling language metamodels for designing features of rich internet applications", *International Journal of Integrated Engineering*, Vol. 10 No. 6.

Wand, Y. and Weber, R. (2002), "Research commentary: information systems and conceptual modeling—a research agenda", *Information Systems Research*, Vol. 13 No. 4, pp. 363–376.

Wang, Z., Sun, C. and Aiello, M. (2021), "Lightweight and Context-aware Modeling of Microservice-based Internet of Things", *2021 IEEE International Conference on Web Services*, IEEE, pp. 282–292.

Whittle, J., Hutchinson, J. and Rouncefield, M. (2014), "The state of practice in model-driven engineering", *IEEE Software*, IEEE, Vol. 31 No. May/June, pp. 79–85.

Wood, K., Lauff, C., Hui, W., Teo, K., Png, S., Swee, A., Collopy, A., *et al.* (2021), *Design Innovation Methodology Handbook–Embedding Design in Organizations*, Design Innovation Programme/Team, Singapore University of Technology and Design-Massachusetts Institute of Technology International Design Centre.

Yu, F., Pasinelli, M. and Brem, A. (2018), "Prototyping in theory and in practice: A study of the similarities and differences between engineers and designers", *Creativity and Innovation Management*, Vol. 27 No. 2, pp. 121–132.