# Breaking the Walls: A Unified Vision on Context-Oriented Software Engineering

Kim Mens
UCL, Belgium

Nicolás Cardozo
Trinity College Dublin, Ireland

Bruno Dumas, Anthony Cleve
UNamur, Belgium

***Abstract*—In this presentation, we present the need for a unified vision on context-oriented software engineering, that reconciles different research areas in computer science, including programming languages, human-computer interaction, and database technology. We identify three dimensions wherein proposed solutions for achieving context orientation could benefit from cross-fertilisation and integration. We present a conceptual and implementation framework that allows for a characterisation of context-oriented systems regardless of these dimensions, serving as a vehicle to discuss some of the challenges involved in reconciling the different dimensions and how to tackle them.**

***Index Terms*—context-oriented software engineering, dynamic software evolution, programming languages, human-computer interaction, databases.**

## I. Introduction

With the advent of new software and hardware technologies, such as mobile and ubiquitous computing, the internet of things, software-as-a-service, cloud computing, big data, and multimodal interfaces, software has become a continuously evolving artefact. The last decade witnessed a clear trend from traditional software systems to software that is ever more aware of, and adaptive to, changing contextual information. This trend is apparent across multiple computer science fields, such as software engineering, programming languages, human-computer interaction and databases, as well as across different subfields of those fields, and across other dimensions such as the different stages of the development process (e.g., analysis, design, or implementation). Existing context-oriented approaches explicitly posit the notions of context and context-specific software variations, as first class entities that can be manipulated and reasoned about. However, the understanding, representation and handling of these concepts differ with the requirements of each domain. Integrating these different individual approaches calls for a holistic vision that unifies the different solutions that have been proposed in the computer science landscape. Some of these fields already start to converge on their view of context orientation. We predict that such cross-fertilisation among fields will increase further in the coming years, driving the computer science community to a new vision on context-oriented software engineering that carries the notion of context at its heart.

## II. Dimensions

Techniques for building context-oriented software are currently being explored independently in a variety of computer science fields. Working towards our goal of a unified vision on context-oriented software engineering, we propose an initial classification of the research space in different dimensions that could benefit from cross-fertilisation, and classify some existing research domains along these dimensions, before discussing cross-fertilisation opportunities.

*a) Dimension 1. Technological perspective.* From a technological perspective, the problem of building context-oriented software systems that can adapt dynamically to context changes has been studied from at least three different angles. Programming language research has explored novel programming paradigms to dynamically adapt the behaviour of running systems according to detected context changes. Database research has studied context-aware database technology and more flexible query languages. Human-computer interfaces have studied the problem from the point of view of user interfaces, including multimodal interfaces. However, building a software system, whether it be mobile, web or of another kind, requires taking into account all of these viewpoints and their interactions together, not in isolation, especially in the light of dynamically changing contexts.

*b) Dimension 2. Software lifecycle phases.* This dimension is concerned with the different phases of the software development lifecycle in which contexts are, or should be, taken into account, ranging all the way from requirements engineering to runtime support. These phases should look into the problem of context orientation in an integrated way, e.g., by agreeing on a unified model for representing and handling contexts and context-specific variations. For example, recent proposals on context variability modeling (at the requirements level) bear close resemblance to how context-oriented programming languages represent and manipulate contexts, dependencies between contexts and context-specific software variations.

*c) Dimension 3. Human perspective.* This perspective, often neglected by software engineers, has been studied actively by sociology researchers, where the notion of context can be seen as a user-centered concept and as an individualising paradigm. This notion raises sociological and ethical questions. In particular, the question of user acceptance is crucial to software engineering, having an important impact on the design and implementation of context-oriented software systems.

## III. Conceptual framework

As a second step towards our goal of achieving a unified vision of context-oriented software engineering, we conceived a conceptual framework, shown in Fig. II, that describes

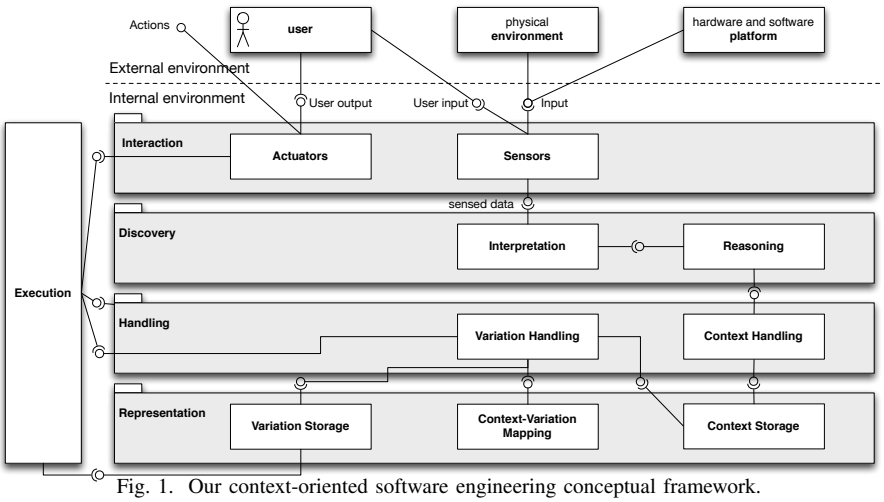

Fig. 1. Our context-oriented software engineering conceptual framework.

the different building blocks composing a context-oriented software system. The framework is generic enough to cover most existing context-oriented approaches targeting different software life cycle phases and artefacts, yet specific enough to discuss in a more focused way how to unify these different approaches, while remaining sufficiently concrete to serve as a possible implementation framework for building context-oriented software systems.

The framework has a layered structure clearly separating the external environment interacting with the system, from how the system interacts with that external world and discovers relevant contexts of use, to how the system handles contexts and context-specific variations, down to how contexts and variations are represented internally, and finally how the actual selection and execution of context-specific variations at runtime are implemented.

## IV. Cross-fertilisation

The layered and modular architecture of the framework facilitates the comparison of, and reasoning about how to unify, different approaches for context orientation. For example, how different approaches compare with regard to how software can vary its behaviour dynamically in reaction to changing contexts, is a question pertaining primarily to the *variation handling* component (together with the *context-variation mapping*, specifying variations-contexts associations.

The technological dimension regards different implementation technologies and how their interaction should be addressed in the representation layer, more specifically in the *variation storage* component. This component defines how context-specific variations are represented and implemented. For example, does each context-specific variation need to address the user interface, database and behaviour aspects jointly, or does each variant focus on a single technological domain only?

Although the framework is software methodology agnostic, the lifecycle phases dimension can be addressed by looking at how to unify different lifecycle activities from the point of view of the framework components. For example, questions regarding how to model context and context dependencies in a unified way pertain to the *context storage* component. Other activities, like verification can be decomposed in subtasks for specific components. While the *context handling* component can verify certain context dependencies dynamically, the *contact storage* component can verify some dependencies statically, and the *variation handling* component can verify interaction validity amid variations and their associated contexts.

The human perspective dimension is apparent in the framework with the explicit notion of users, and user input/output in the interaction layer. But this dimension affects other components too. For example, one way of dealing with user acceptance is by providing gentle transitions, that help users be aware of new active variations upon context changes. What components would be in charge of representing and handling such transitions? The framework provides a structured frame of reference, as well as an implementation architecture, to reason about such key issues.

## V. Conclusion

Significant advances have been made in different fields of software engineering, to the domain of context orientation, which studies the ability of software systems to vary their behaviour dynamically to changing contexts of use. These fields have mostly studied the problem in isolation, thus missing important pieces of the puzzle to build context-oriented software systems that can fully adapt to changing contexts. This observation calls for a unified approach that integrates all of these contributions, consolidating these research fields, and raising users' acceptance of autonomous and adaptive software systems as part of their every day life. Towards achieving this goal, we proposed a conceptual and implementation framework as a research vehicle to compare and integrate different existing approaches and to actually implement a prototype of a unified context-oriented development environment.