Information and Software Technology xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Synchronised visualisation of software process and product artefacts: Concept, design and prototype implementation

Mujtaba Alshakhouri, Jim Buchan, Stephen G. MacDonell*

SERL, School of Engineering, Computer and Mathematical Sciences, Auckland University of Technology, Private Bag 92006, Auckland 1142, New Zealand

ARTICLE INFO

Keywords:
Software visualisation
Conceptual visualisation
Software process
Conceptual design
Feature location
Traceability, locality

ABSTRACT

Context: Most prior software visualisation (SV) research has focused primarily on making aspects of intangible *software product* artefacts more evident. While undoubtedly useful, this focus has meant that *software process* visualisation has received far less attention.

Objective: This paper presents *Conceptual Visualisation,* a novel SV approach that builds on the well-known CodeCity metaphor by situating software code artefacts alongside their software development processes, in order to link and synchronise these typically separate components.

Method: While the majority of prior SV research has focused on re-presenting what is already available in the code (i.e., the implementation) or information derived from it (i.e., various metrics), the presented approach instead makes the design concepts and original developers' intentions – both significant sources of information in terms of software development and maintenance – readily and contextually available in a visualisation environment that tightly integrates the code artefacts with their original functional requirements and development activity.

Results: Our approach has been implemented in a prototype tool called ScrumCity with the proof of concept being demonstrated using six real-world open source systems. A preliminary case study has further been carried out with real world data.

Conclusion: Conceptual Visualisation, as implemented in ScrumCity, shows early promise in enabling developers and managers (and potentially other stakeholders) to traverse and explore multiple aspects of software product and process artefacts in a synchronised manner, achieving traceability between the two.

1. Introduction

In spite of its comparatively young age, the sub-field of Software Visualisation (SV) has advanced rapidly during the past two decades, with a proliferation of new research being published particularly during the last fifteen years. Much of this research has been highly innovative. Its value to the software engineering (SE) community — particularly in promoting comprehension and awareness — has been substantial, and the field has become well established in the research literature [1].

That said, the range of problems explored in recent SV research has been limited. Not unexpectedly, software visualisation research has been concerned with redressing the intangible nature of software, and this has naturally led researchers to direct their attention toward visualising (aspects of) software product artefacts; by comparison, however, far less attention has been afforded to other facets of software development. In fact, it could be argued that the three major categories of SV that have shaped much of existing SV research, namely visualisation of software static structure, visualisation of program runtime behaviour, and visualisation of software evolution [2], have inadvertently played a role in limiting the field's growth and expansion in considering other important aspects of software. It is our particular contention that contemporary visualisation technologies have the potential to also make visible numerous aspects of the *software development process* that are equally disadvantaged by the intangible nature of the end products (i.e., the software artefacts). Extending the benefits of visualisation to important aspects of development is expected to contribute in rendering these technologies as valuable for an even wider range of software practitioners and practices.

Such concerns have been raised by researchers previously. For example, Storey, Čubranić, and German, in their 2005 paper, called for visualisation techniques that provide *activity awareness* [3]. Similarly, Petre and de Quincey highlighted the need to represent *design concepts* in software visualisation approaches in their 2006 paper [4]. Our review of the recent literature (detailed below), however, shows that the software development process is still not being addressed. While there is SV research that highlights some *effects* of the software process on

* Corresponding author.
E-mail addresses: malshakh@aut.ac.nz (M. Alshakhouri), jim.buchan@aut.ac.nz (J. Buchan), stephen.macdonell@aut.ac.nz, stevemac@acm.org (S.G. MacDonell).

https://doi.org/10.1016/j.infsof.2018.01.008
Received 29 May 2017; Received in revised form 15 January 2018; Accepted 19 January 2018

software artefacts, the majority of it is still directed to approaches and techniques for visualising the product. It is our assertion that visualising the software process *in its own right* not only has several potentially beneficial implications for the software industry, but also the wider stakeholder group. For example, from a cognitive perspective, visualising processes *in the context of* software artefact structure, and vice versa, could be important in terms of increasing stakeholder awareness and understanding of *both* the processes and their implemented product artefacts.

In this paper we address this untapped potential of the SV discipline to benefit the software development process by introducing a novel visualisation approach that firstly captures significant aspects of the development process, in our case Scrum, and then tightly integrates and synchronises these with the product artefacts that are created by it. Inspired by the early work of Petre and de Quincey [4,5], we call this new approach *Conceptual Visualisation*. The approach enables the visual presentation of user requirements as well as developers' design concepts directly 'over' the visualisation scene, seamlessly linked to the actual implementation of those concepts and requirements. It is expected that this new visualisation technique will better inform and support several software tasks and activities, for example, concept location. The approach should also bring software visualisation technologies closer to practitioners through a range of potential applications of use by a variety of software stakeholders. Example applications include: bidirectional requirements traceability, feature (or concept) searching, development progress monitoring, support of group design reasoning, and improved stakeholder communication. Section 6 provides some concrete examples of such usage scenarios. A proof-of-concept prototype tool that implements this new visualisation concept has been developed in order to demonstrate its viability as well as to assess its potential utility for practitioners.

The rest of this paper is structured as follows. Section 2 discusses the status quo of the software development process in the SV literature to situate the research in this paper. Section 3 reflects on prior literature that has emphasized the need to equip visualisation techniques with a more useful, or at least broader, set of information, and in doing so, provides the context for introducing our *Conceptual Visualisation* approach. Section 4 presents our proposed visualisation technique and describes its key technical details along with the major functional features it facilitates. In Section 5 we cover the tool building and a proof of concept demonstration of the new approach. A preliminary evaluation is then presented in Section 6 followed by some reflections on the approach and its implications for practice in Section 7. Some final concluding remarks are made in Section 8.

2. Software development processes in SV research

As noted in Section 1, the issue of SV research being confined to the consideration of a relatively narrow range of SE aspects has been discussed from different perspectives in a range of prior research. However, it remains evident in recent literature that the response to such discussions seems to have been rather sparse, with the majority of SV research to date being primarily oriented toward re-presenting the product and relatively little addressing the process. To the best of our knowledge, only two forms of SV research address aspects of the software process: a limited research stream on *human activity awareness*, and, more recently, work on the visualisation of *social network analysis*. Interestingly, this dearth of SV research into process appears to be specific to the field of SE: Petre and de Quincey [4] note that in other scientific and engineering disciplines, it is in fact the *development process* that is primarily addressed by visualisation technologies. An explanation for this limited treatment of the software process by the SV community compared to other domains could be that in these other domains it is the process that is intangible, whereas in SE *both the emergent product and its development process* are intangible.

While the application of SV to the software process has been relatively minimal, it has been extensively applied to the software product. Some researchers have gone as far as to suggest that there is currently an abundance of techniques for visualising software artefacts themselves, but that there is a lack of techniques to address other important aspects of software, or in the design of appropriate visual metaphors to incorporate those aspects alongside the visualised artefacts [3]. It remains that both the software static structure and software evolution visualisation categories have rarely considered aspects of the software development process. It is contended here that development processes carry important information that is potentially valuable for various software tasks but that are commonly not documented or have documentation that has no low-effort, straightforward mechanism for software engineers to link to the source code. We address this with the development of the new tool and approach described in this paper. To the best of our knowledge, there is currently no single tool or approach like this that considers the presentation of the software development process as captured by design concepts *in the context of* the software structure, and vice versa. This new approach we describe presents software code artefacts alongside their development processes directly in the visualisation scene.

In one of the first studies to raise the lack of attention to software process in SV, Storey et al. [3] emphasize the need to promote human activity awareness in SV tools, emphasizing its central importance to its practical utility in answering many relevant questions for software stakeholders. After exploring several SV tools, however, they concluded that only a few offered reasonable support for human activity awareness. In the years since, a handful of new approaches have appeared that do indeed attempt to support some forms of activity awareness in their visualisation techniques, most notably: Manhattan [6], StarGate [7], code_swarm [8], and more recently, Replay [9]. Human activity, however, is an *effect* of the actual development process. Soon after the Storey et al. work appeared, Petre and de Quincey [4] signalled that it was the missing *development process* that should be the focal point behind promoting awareness, and referred to the elements of awareness discussed by Storey and colleagues as being only '*subtle*' aspects of software awareness that are a consequence of attention to software change.

Moreover, in examining the literature, it is evident that the tools that have attempted to support activity awareness or have visualised aspects of development have almost all relied primarily on information extracted from IDEs and version control systems. This includes visualisations via heatmaps [10], action graphs [11], and social network analysis [12]. It is similarly evident, however, that the information made available by such tools (and hence the knowledge that could be represented) is limited in nature and is generally only commit-based data. It does not capture design concepts and is confined instead to modification activities. In fact, the authors of the Replay tool have specifically stated that (p.755) "… *the coarse-grained nature of the data stored by commit-based software configuration management systems often makes it challenging for a developer to search for an answer.*" Another issue common of these tools is that, apart from the Manhattan tool, they do not present the extracted data in the context of the actual software structure. From a cognitive perspective (discussed in the next section) this additional functionality could play an important role in supporting a range of software tasks [10].

We argue that by finding a mechanism to represent the core aspects of the software development process *within* the context of software structure visualisation, important questions that software engineers, developers, managers, and customers (and possibly other stakeholders) might pose could be more readily answered. This includes almost all the activity awareness questions discussed previously [3,4] and that fundamentally revolve around **authorship, rationale, time**, and the **artefacts** themselves. The next section presents and discusses this mechanism.