Extending Software Systems While Keeping Conceptual Integrity

Reuven Yagel
Software Engineering Department
Azrieli – Jerusalem College of Engineering
Israel
robi@jce.ac.il

Abstract— **Design and analysis of software systems in terms of their Conceptual Integrity is a demanding task. Nonetheless, progress has been made in recent years and actual software systems in practical use, such as Git, have been analyzed. In this work we make a further first step within the conceptual analysis approach, by asking how to extend software systems by addition of further components while keeping Conceptual Integrity of the resulting system. We propose specific techniques to this end. As a case study to illustrate these techniques, we analyze a popular project management service, namely Gitlab, for its various services integrity and adaptability to software engineering lifecycle stages.**

Keywords- Component; Software Repository; Lifecycle Management; Conceptual Integrity; Git; Gitlab

I. INTRODUCTION

Conceptual Integrity has been identified by Brooks [1, 2] as the main challenge for software system design. More recently, Jackson et. al. [3, 4] further formulated and demonstrated the Conceptual design and analysis approach for a popular software system – git [5]. In this paper we build on this work and start broadening this view by examining the further issue of extending software systems with additional components while preserving the Conceptual Integrity of the whole system. We propose some specific techniques in this respect and illustrate them by analyzing popular code repository and project management services.

A. Git Based Software Project Management Systems

The software development industry is going through major changes in recent years. Among them are new tools and services for software project management (SPM) such as github.com [6]. This service, taken as an example, is based on cloud hosting of git – a distributed version control system. Beyond managing source code versions, the service simplifies workflows of branching and merging and also includes various project management tools, in particular social features, which allow vast collaboration options. A variety of software organizations and their offered services, have different service models. Examples of the referred kind are SourceForge, Microsoft's Team Foundation Services, bitbucket, Coding.net and many others.

In this work we focus on such a similar service – gitlab.com [8]. It is also a software project management service centered on git. Beyond managing source code versions, it includes various project management tools for communication, documentation, testing and more.

The reason for choosing git-centered software systems to illustrate our proposal, is to capitalize on existing analyses of Git and Gitless by Jackson and co-workers [3], [4], and their wide industry usage and acceptance.

B. Paper Organization

The remainder of this paper is organized as follows. In section II we review related literature. In section III we describe a general Lifecycle Model underlying the software systems used to illustrate our approach. In section IV, specific techniques of our approach to extend software systems while keeping conceptual integrity are proposed. In section V the Gitlab software system serves to illustrate our proposed approach. The paper is concluded with a short discussion in section VI.

II. RELATED LITERATURE

Brooks [1, 2] suggested three principles for representing the notion of conceptual integrity:

- Orthogonality
- Propriety
- Generality

Jackson et. al. [3] showed how git conceptual design is quite complex and also problematic in light of these principles. They go further on and suggest a new design, titled gitless to correct those issues.

Git [5] is an open source distributed version control system. It became quite popular in recent years, but still criticized for its usability and conceptual integrity issues.

Models of software development as well as processes, methods, and tools are widely discussed. Here we reference mainly Rajlich [8], but many others are discussing those, e.g., [11-15].

III. LIFECYCLE MODEL

Since the discussed services are all aimed for helping developing large software projects, we choose here a general

DOI: 10.18293/SEKE2017-205

model for software project lifecycle adapted from Rajlich [8]. It is a basic and general staged model of software "lifespan", which includes the stages of:

1. *Initial development,*
2. *Evolution,*
3. *Maintenance* (servicing),
4. *Phase-out,*
5. *Close-down.*

The second and third stages are usually iterative and incremental (see Figure 1 of [8]). A Software Project Management System is expected to support development based on such a model and its various derivatives or alternatives.

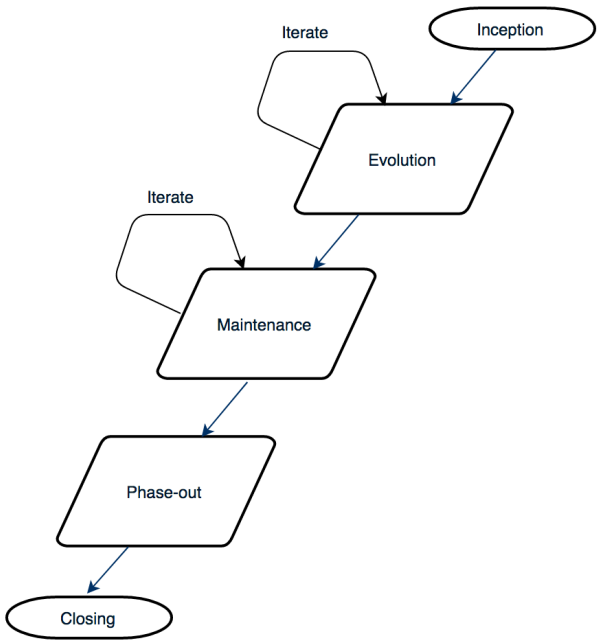

Figure 1 Staged model of software adapted from [8]

IV. Our Approach: Software Extension with Conceptual Integrity Preservation

Our approach consists of enabling addition of services along the lifecycle of a software system, while having an infrastructure to preserve Conceptual Integrity.

Typically, most SPMs contain at least the following services:

1) Version Control for code/software (SCM)
2) Documentation system
3) Issue Database
4) API and integration points with other services, e.g., continuous integration (CI).

We now formulate our expectations from a SPM system in the light of the above mentioned conceptual integrity concepts. In other words, can we forecast eventual Conceptual Integrity violations from these typical services?

- *Orthogonality* – in principle each of the above services should be used standalone, although in practice users will expect integration between them. As an example: there might be an open issue to fix some missing documentation. Upon update of the documentation (2), the version control system is preserving a commit (1), the issue changes status to closed (3) and a CI service is triggered to run tests (4).

Here is the analysis:

- *Propriety* – the set of services above is quite concise and centered around project management. Extensions are mainly through 3rd party software integrations.
- *Generality* – each service can be adapted to a specific project lifecycle and the actual project being carried. For example, version control is not limited to a specific programming language (or even software artifacts at all). The use of git, opens the door to many possible git branching models [9].

From this preliminary analysis, the requirements for a Conceptual Integrity preservation infrastructure are formulated as specific design patterns for Conceptual Integrity to be prepared in advance:

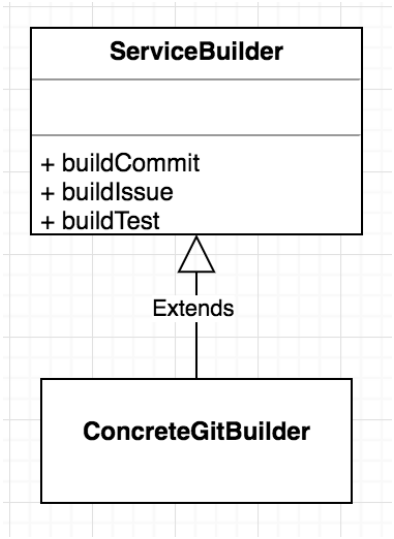

Figure 2: Using Builder Concatenation Pattern

Concatenation patterns – design patterns that can be instantiated for concatenation of services, as in the above example, which illustrates the Orthogonality principle. These patterns could use

for example the "builder" GoF [10] pattern (shown in Fig. 2), suitable for setting up a related set of services.

Integration interfaces – this set of patterns should act as a sort of "adapter" or "façade" GoF patterns, with the purpose of integrating yet unknown 3rd party software.

Models and languages variations – these patterns needed to cope with variation of programming languages and say branching models, should resemble GoF "strategy" patterns.

V. The Gitlab Case Study

We shortly discuss the Gitlab services, and their conformance to conceptual integrity principles. The Gitlab feature page [7] presents the software as containing "Powerful features for modern software development, tightly integrated into one platform". It is centered around a hosted git server with a rich web interface.

- *Version Control* – Gitlab suggests hosting of unlimited private or public git repositories, thus conforming to generality. The web platform is used to provide accessible and convenient graphical user interfaces for the various version control operations. Gitlab can be installed on premise or consumed as a cloud service. Still the usability issues of git are not totally masked, and this might explain the slow adoption rate of such a service outside of the software engineering community.
  Gitlab has a snippet service for sharing portions of code. Such a service could be attitude as an extension to version control; alas a snippet is not versioned!
  Gitlab added fined grain access control (FGAC) and several privacy tracks, which in fact are evidently options for version control especially for enterprise settings. Specific files or folders can be locked in order to prevent merge conflicts. Gitlab also suggest sharing small portions of code by a code snippet service.
- *Documentation* – Gitlab provides a wiki system and also a static site hosting solution (Gitlab pages). The wiki system is maintained in a separate and less accessible git repository instance. This is, in our mind, a conceptual integrity issue, since project artifacts are handled differently (github has the same issue). Also, it is a common practice to have a major documentation file – usually named *Readme*, which resides in the main source tree. Thus, project documentation become spread in at least three different areas – source code, wiki and a static site, not to mention other possible documentations (formal or informal).
- *Issue Database* – the text of an issue can link to other artifacts, and there are ways to change issue statuses from version control commit messages. This is another potential point were conceptual integrity can be weakened. Gitlab also has an activity stream, so users need to adjust their most efficient project update communication patterns.
- *API and integrations with other services, e.g., continuous integration (CI)* – some of the services are hosted and some are just interfaces to 3rd parties. An infrastructure can help define the interfaces in more concise ways to improve conceptual integrity.

VI. Discussion and Future Work

In this position paper, we outlined an ongoing research to examine the conceptual integrity of complex software systems and laid directions for patterns for conforming to integrity principles. We intend to further analyze in detail Gitlab and compare it to other services. The expected outcome of the analysis and comparisons with other services is a suggested series of concrete detailed improvements of the system.

The economics of the competition between companies suggesting those services might lead to "feature creep" followed by breaking conceptual integrity. This is indirectly demonstrated in the long feature lists in the providers' websites.

Conceptual integrity will also be discussed from other angles, e.g., operations, hosting options, and pricing models.

Acknowledgment: I would like to thank Iaakov Exman for helpful and fruitful advice.

References

[1] F.P. Brooks, *The Mythical Man-Month – Essays in Software Engineering* – Anniversary Edition, Addison-Wesley, Boston, MA, USA, 1995.

[2] F.P. Brooks, *The Design of Design: Essays from a Computer Scientist*, Addison-Wesley, Boston, MA, USA, 2010.

[3] D. Jackson and S. P. DeRosso, "What's wrong with git?: a conceptual design analysis" in Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming & software (Onward!), 2013.

[4] D. Jackson, "Towards a Theory of Conceptual Design for Software", in Proc. Onward! 2015 ACM Int. Symposium on New Ideas, New Paradigms and Reflections on Programming and Software, pp. 282-296, 2015.

[5] S. Chacon, Pro Git, Apress 2009, http://git-scm.com/book

[6] Github, https://github.com/about

[7] Gitlab, https://about.gitlab.com/features/

[8] V. Rajlich, "Software Evolution and Maintenance", in Proceedings of the on Future of Software Engineering (FOSE), 2014.

[9] Atlassian Tutorial, "Comparing Workflows", https://www.atlassian.com/git/tutorials/comparing-workflows

[10] E. Gamma et. al., Design Patterns: Elements of Reusable Object-Oriented Software, AddisonWesley Professional, 1994.

[11] Amber, The Agile System Development Life Cycle (SDLC), http://www.ambysoft.com/essays/agileLifecycle.html

[12] P. A. Laplante, What Every Engineer Should Know about Software Engineering, CRC Press, 2007.

[13] R. S. Pressman, Software Engineering: A Practitioner's Approach, 8th Edition, McGraw-Hill Education, 2014.

[14] B. Boehm, "A Spiral Model of Software Development and Enhancement". In: ACM SIGSOFT Software Engineering Notes (ACM) 11(4):14-24, 1986.

[15] S. McConnell, "7: Lifecycle Planning", in Rapid Development: Taming Wild Software Schedules. Microsoft Press., 1996.