FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

Arquitectura de Sistemas de Software

28th April 2015 • Personal notes allowed • Duration: 75+15 minutes

Please carefully read the description of the software system below and answer the questions **always justifying them succinctly and clearly**, eventually mentioning the bibliography or references that fundament them. When needed, you should explicit all the assumptions you did to answer the questions.

Weakly Typed Wiki (aka **WEAKI**) is a new kind of wiki that allow developers to easily formalize their software knowledge, in a way that is "precise as needed", varying from free text to very structured text or code. It combines the best of two worlds: free text wikis with semantic wikis.

WEAKI provides an easy web access to contents stored in GIT repositories, private and public, depending on the permissions of the user. WEAKI is structured in the following subsystems, below described in detail: web interface, contents editors, storage system, type engine, and team collaboration.

1. From a global perspective, which key architectural styles (2-3) do you see as helpful to design the overall system, and why. Please explain by identifying some examples of components and connectors of WEAKI that may instantiate those styles.

   R: repository (5), layered (2), interpreter (2), event-driven (1).

2. Considering the design patterns you studied (mainly GoF), suggest a **partial class diagram** for each one of the issues below and **justify** the patterns (if any) that you think are most appropriate to solve it.

   a. WEAKI must support the notion of software project, teams of developers, eventually aggregating sub-teams, and so on, in a nested way. The access to contents should be managed by the GIT authentication and authorization system, which may have similar concepts but not exactly the same. How to interoperate the WEAKI with GITHUB notions of repositories, collaborators, and organizations?

      R: abstract factory (6), composite (4), adapter (3), mediator (4)

   b. Software contents have different types (code, models, text) and although independently stored, they may have implicit dependencies between them that must be managed in order to propagate their needs for change. For example, when a fragment of code is included in a page, the page and its authors should be notified when that specific fragment of code changes, so that everybody involved can decide whether to change, or not, something in that page. How to cope with this need?

      R: observer (7), mvc (5), chain of responsibility (2)

   c. In order to be able to include code in WEAKI pages, it must be parsed and formatted. Since code can be written in different programming languages, WEAKI must be able to plug-in different language parsers, and different formatting styles for each language. How to design this specific part of WEAKI?

      R: interpreter (6), template method (3), visitor (2)

3. Focus now in one of the subsystems of WEAKI and identify three features that might need to vary in concrete implementations and therefore should be designed in a way that is easy to adapt without much effort. For each feature, describe it succinctly, its variability needs, and name the design patterns that you have in mind to apply.

   R: feature (2), variability (1) x 3 + 1

Note: each question has a value of 15% and the global evaluation values 10%.

The End.