Note: This information may still be changed or updated during the course.
Last update: February 23, 2010

Software Architecture Project 2010

This term you will carry out a project with the prime goal of putting to practice the main principles and techniques learned in the course Software Architecture. You can choose one of the project ideas described below:

HTML document library

Many applications generate output, such as testing statistics or simulation results, which can be reported to the user in static HTML format. However, generating such static websites is often messy, especially when HTML is handled as strings in the code. Hours can be spent on debugging wrong links, forgotten closing tags, and hard-to-read indentation.

The goal of this project is to relieve the programmer from this burden. A library of Eiffel classes will handle URL generation and linking, correct tag naming and indentation, writing well-formed HTML, etc. The library's API should support creating a website with multiple linked pages with images, bullet lists, tables, etc. It should also be possible to include existing HTML snippets in a document, e.g. an image map already stored in a file or in a string object.

The architecture of the library is of course left to your creative imagination and pragmatic instincts, but make sure that it is extensible for other output formats. Some design patterns you might consider include the visitor, composite, builder and decorator. As you will see during the course, different techniques and choices can impact the code's maintainability, flexibility to accommodate certain features and the ease with which others can use it.

Data Structure Visualizer

A nice way to understand data structures is to see it in a graph, which reveals the values of objects and their connection relationship. For example, compared to checking object fields, a two-column table is a more direct way to review a HASH_TABLE object.

The goal of this project is to design a data structure visualization library. The library can be used able to analyze a given set of Eiffel objects, and visualize these objects in certain formats. You should consider design principles and design patterns learned from the course to achieve flexibility in the following dimensions:

- The library should handle arbitrary Eiffel objects.
- There should be special support for some important classes, such as ARRAY, LIST and HASH_TABLE.
- Information which represents an object.
- The visualization can be rendered through different means, for example, through UML, dot format, or through EiffelStudio's diagram tool.
- Mechanisms to handle objects with large size. A typical question is: how to display a list with a thousand elements?

The Software Architecture project is a team effort. Within the next few days, you should form groups of at most four people. Within the team you must negotiate how much and what each person will contribute. Try to divide the work equally among the team members and start thinking about when and where you will do the work. We have planned to devote every week at least one of the three hours of the Software Architecture exercise sessions (Monday, 15-18) to the project, so you will be able to discuss and work on the project during this time.

The project will be graded and contributes 50% of the final course grade. To help you plan your workload during the semester, we have adopted a project structure with six phases. Each phase has a strict deadline and deliverables that need to be handed in. Additionally to the deliverables, for some of the phases you will need to give a short presentation in the exercise session following a deadline. The presentation should summarize what you have done in the past project phase. Below you find the deadlines, descriptions of the deliverables, and grading criteria for each of the phases.

Your project should be hosted on Origo (http://origo.ethz.ch). All the deliverables of the project steps will be submitted by sending an e-mail to your assistant containing links to the according wiki pages and SVN directories including revision numbers or tag names where applicable. We have prepared an example project on Origo found under http://soft-arch-project.origo.ethz.ch. It shows how we would like you to structure your project and contains templates for some of the project deliverables.

To setup your own Origo project follow these steps:

1. Every member of your team needs to create an Origo user: http://www.origo.ethz.ch/user/register (unless she/he already has one). We propose that you use your NETHZ account name as Origo username.
2. Decide on a name for your project. It should be prefixed with "sa10-", e.g. "sa10-my-project".
3. One of the team members then has to request a new project at http://www.origo.ethz.ch/origo_home/create_project. The **Project Type** should be "Closed Source" and **Project Listing Visibility** should be "Not Visible". After having done this, you will have to wait until the Origo team approves of your request and sends you an e-mail that the project has been created.
4. Once you have received a confirmation that your Origo project was created, you go to your project's Origo page *http://[your-project-name].origo.ethz.ch* and sign in with your username and password. On the left side of the page, you find a navigation menu with an entry "Project". Clicking on it will display a list of project owners and members. Add your team members to the project, so that they can modify the wiki pages and commit code to SVN. Also add your assistant as a member of the project.
5. Set the structure of your Origo project up to contain the following directories and documents:

```
Subversion:
$ROOT = https://svn.origo.ethz.ch/[your-project-name]
$ROOT/documentation - Documentation
$ROOT/presentation - Presentation slides
$ROOT/src - Source code
$ROOT/test - Test suite

Wiki pages (all set to "Private", i.e. only visible to the developers of a project):
$ROOT = http://[your-project-name].origo.ethz.ch
$ROOT/wiki/requirements
$ROOT/wiki/design
$ROOT/wiki/test_plan
$ROOT/wiki/test_report
```

Additionally, you will need to install some software (if you haven't installed it already). For working with SVN repositories under Windows, we recommend TortoiseSVN (http://tortoisesvn.net/downloads). For the development of Eiffel projects, you will need EiffelStudio (use version 6.4 or later available from sourceforge http://sourceforge.net/projects/eiffelstudio/).

A word on the deliverables: The project should be **closed source**, which means that only team members can read (and write) the SVN repository. Additionally, you will have to hand in several documents. We provide templates in the form of wiki pages, but you may choose another text format (e.g. PDF). If you choose another format then you will have to commit the documents into the "document" folder on SVN and write on the wiki page where the document can be found. If you choose to write the documents as wiki pages, then set them to "Private". This can be done on the editing interface of the wiki pages in Origo. Note that simultaneous editing of a wiki page is not possible.

1. Requirements specification

Handout: Monday, 01.03.10
Deadline: Sunday, 21.03.10
Presentation: Monday, 22.03.10
Time: 20 days

The ultimate purpose of requirements specification is to describe *what* qualities a system must exhibit and *what* goals it must reach. It does not describe *how* these qualities and goals are achieved; this is left to the engineer. It does, however, specify the interfaces between the system and its external environment (which can have physical and logical components) and decide on the relevant stakeholders of a project. The result of the requirements engineering phase of a project is a **software requirements specification (SRS)** capturing stakeholders' expectations and providing a basis to enable software engineers to develop a solution that will satisfy the requirements.

Your task:

Your task is to develop a SRS document for your project. We provide a template (see http://soft-arch-project.origo.ethz.ch/wiki/requirements) containing the structure and sections that probably are relevant to your project, but you may add and remove sections as you see fit. The SRS document will serve to clarify the goals of your chosen project and prioritize the implementation of functionality. Since your customer is the assistant, you should confer with him if you need any clarifications. Experience has shown that the SRS developed in such a course project is usually about 10 to 20 pages long.

The presentation should give an overview of your SRS and explain the most important points. If there were any issues during the SRS writing phase that created discussion among the team members, they could make an interesting presentation topic. The presentation should be between 5 and 10 minutes long (= 4-8 slides).

Resources:

- SRS template page (http://soft-arch-project.origo.ethz.ch/wiki/requirements)
- SRS standard IEEE Std 830-1998 (http://ieeexplore.ieee.org/iel4/5841/15571/00720574.pdf -- download for free from within ETH network)
- Directory structure for SVN (as shown under https://svn.origo.ethz.ch/soft-arch-project/)

Deliverables:

- SRS document available at or linked from *http://[your-project-name].origo.ethz.ch/wiki/requirements* (only available to members of the project).
- Slides of your presentation on SVN available at *https://svn.origo.ethz.ch/[your-project-name]/presentation*.

Grading criteria:

We will use the following grading scheme for your SRS with a maximum number of 40 Points.

Readability and structure of the SRS	**0: totally disagree, ..., 4: totally agree**
The document exhibits a clear structure and contains all needed and only relevant sections (e.g. title, names of authors, date, purpose, scope, ...).	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The text reads well (i.e. it is precise, concise, and graphically and grammatically correct).	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
Relevant technical terms and abbreviations are defined (e.g. in the glossary) and used consistently throughout the document.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
Quality of the requirements	
Functional requirements are associated with justified testable quality measures.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
Non-functional requirements are associated with testable, justified, and relevant quality measures.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
Functional requirements are associated with reasonable priorities and non-functional requirements are either prioritized or an alternative plan is given.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The functional requirements are complete, in the sense that no important requirement is missing.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The functional requirements are relevant and support the stated goals.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The functional requirements describe a useful system.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
Presentation	
The presentation covered the required topics.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]

2. API design

Deadline: Sunday, 11.04.10
Presentation: Monday, 12.04.10
Time: 21 days

In the design phase of a project, the structure of a software system is developed on several levels of detail. The software requirements feed the design task. The software design specification focuses on *how* the system will be constructed. It includes four models: the data design describes all data structures to be used, the architectural design describes the overall system, the interface design develops the user interfaces such as GUIs, and the component-level design specifies the components (classes) of a system.

Your task:

Design a system that satisfies the requirements specified in the SRS. Decide on how to break down your system into components and how they are composed. Think about possible applications of design patterns, choices for data structures, and possible extensions to your system. While developing your software design, keep in mind that we expect you to critically analyze your choices. This will be done in several sections of a design document (see http://soft-arch-project.origo.ethz.ch/wiki/design).

The deliverables of this project step are a set of Eiffel classes (with feature declarations and contracts, but empty or deferred feature bodies) and a design document describing your choices and giving an overview of your system architecture. To generate diagrams for the document, you may use EiffelStudio's diagram tool. Make sure that the Eiffel classes allow writing test cases (as performed in the next project step) and make sure that you provide stable interfaces. Note that if you need to change the API later in the developing phase, it is your responsibility to renegotiate with the testing team.

In the presentation you should show an overview of your design. Explain important design decisions and describe which patterns you used. The presentation should be between 5 and 10 minutes long (= 4-8 slides).

If you want feedback on your design before the deadline, hand in a draft to your assistant **before Tuesday, 30.03.10**.

Resources:

- Design document template (http://soft-arch-project.origo.ethz.ch/wiki/design)
- Directory structure for SVN (as shown under https://svn.origo.ethz.ch/soft-arch-project/)
- Your requirements document of step 1 (*http://[your-project-name].origo.ethz.ch/wiki/requirements*)

Deliverables:

- Set of Eiffel classes on SVN available at *https://svn.origo.ethz.ch/[your-project-name]/src* (tell your assistant the revision number or tag name).

- Design specification document available at or linked from *http:///[your-project-name].origo.ethz.ch/wiki/design* (only available to members of the project).
- Slides of your presentation on SVN available at *https://svn.origo.ethz.ch/[your-project-name]/presentation*.

Grading criteria:

We will use the following grading scheme for your design with a maximum number of 40 Points.

Design document	**0: totally disagree, ..., 4: totally agree**
The design document exhibits a clear structure and contains all needed and only relevant sections (e.g. title, names of authors, date, purpose, scope ...).	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The design document convincingly discusses the reasons for the choice of design patterns.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The design discussion convincingly describes the benefits and drawbacks of the chosen design.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
Code base	
The code base is cleanly designed (e.g. uses appropriate grouping of features and classes, consistent naming conventions, meaningful class and feature comments).	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The classes include relevant contracts.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
General design	
The choices of design patterns are adequate and justified.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The design satisfies the requirements of the SRS and reaches the goals.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The design is extensible, reusable, and efficient.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The overall impression of the design is excellent.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
Presentation	
The presentation covered the required topics.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]

3. Test plan

Deadline: Sunday, 25.04.10
Presentation: Monday, 26.04.10
Time: 14 days

In this project step, you will receive the SRS and the design document, and the Eiffel classes of another team's project. Based on this material you develop a test plan and a test suite for their project. This means that you will take the role of external test engineers.

Your task:

Get familiar with the project of the other team by reading the provided documentation and exploring the Eiffel classes. Feel free to ask them any questions. Your goal as test engineers is to develop a strategy and test cases that reveal as many bugs as possible in the implementation (that is yet to come in the next project step). To do this, first make a list of test items and decide which features of the software should be tested and which should not be tested. Also think about criteria for considering a test case to pass or fail. The template for the test plan is available at http://soft-arch-project.origo.ethz.ch/wiki/test_plan. Then develop the according test cases and put them into your "test" directory on SVN.

The presentation for the test plan should explain your strategy of selecting which features to test and what kind of test cases you developed. Describe why you think that your test cases are sufficient. If you do not test certain parts of the system, explain why. The presentation should be between 5 and 10 minutes long (= 4-8 slides).

Resources:

- Test plan template (http://soft-arch-project.origo.ethz.ch/wiki/test_plan)
- Tested project's requirements specification (*http://[**other-project**].origo.ethz.ch/wiki/requirements*)
- Tested project's design document (*http://[**other-project**].origo.ethz.ch/wiki/design*)
- Tested project's source code (*https://svn.origo.ethz.ch/[**other-project**]/src*)

Deliverables:

- Test plan available at or linked from *http://[your-project-name].origo.ethz.ch/wiki/test_plan* (only available to members of the project).
- Test suite on SVN available at *https://svn.origo.ethz.ch/[your-project-name]/test* (tell your assistant the revision number or tag name).
- Slides of your presentation on SVN available at *https://svn.origo.ethz.ch/[your-project-name]/presentation*.

Grading criteria:

We will use the following grading scheme for your test plan and test suite with a maximum number of 28 Points.

Test plan	**0: totally disagree, ..., 4: totally agree**
The test plan exhibits a clear structure and contains all needed and only relevant sections (e.g. title, names of authors, date ...).	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The test plan covers the functionality described in the requirements document. If certain functionality is not covered, the test plan explains why (e.g. not testable or hard to test, because...).	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
Test suite	
The test suite covers the most important aspects of the software under test.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The test suite matches the interfaces.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The test cases in the suite covers single function points. In order words, the test cases should be unit test cases.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
Test inputs represent typical usage of the tested project.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
Presentation	
The presentation covered the required topics.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]

4. Implementation

Deadline: Sunday, 16.05.10
Presentation: Monday, 17.05.10
Time: 21 days

After having completed the design, you now need to implement the functionality, i.e. provide the code for the feature bodies.

Your task:

Use the prioritized functional requirements to decide on what functionality should be implemented first. Make sure that your code satisfies both the functional and the non-functional requirements. Additionally, extend the design document with a section on any updates or changes that were needed and the reasons for them. Also, write a paragraph or two on how well you found your design to be for coding the functionality and any problems that arose. The design document contains two sections for this: "Post-design changes" and "Post-implementation reflection" (see http://soft-arch-project.origo.ethz.ch/wiki/design#Post-design_changes and http://soft-arch-project.origo.ethz.ch/wiki/design#Post-implementation_reflection).

In case parts of the code need to be changed, be careful not to destabilize the interfaces that you provided to the test engineers.

In the presentation you should describe if you had any problem during implementation. If you had to revise the design, explain the changes made. If you have an interesting use case of a design pattern or algorithm, show it as well. You can also run a short demo of the project. The presentation should be between 5 and 10 minutes long (= 4-8 slides).

Resources:

- Design document template (http://soft-arch-project.origo.ethz.ch/wiki/design)
- Design specification document of step 2 (*http://[your-project-name].origo.ethz.ch/wiki/design*)
- Source code of step 2 (*https://svn.origo.ethz.ch/[your-project-name]/src*)

Deliverables:

- Set of Eiffel classes on SVN available at *https://svn.origo.ethz.ch/[your-project-name]/src* (tell your assistant the revision number or tag name).
- Design document (extended with two new sections and any other updated material marked) available at or linked from *http://[your-project-name].origo.ethz.ch/wiki/design* (only available to members of the project).
- Slides of your presentation on SVN available at *https://svn.origo.ethz.ch/[your-project-name]/presentation*.

Grading criteria:

We will use the following grading scheme for your implementation with a maximum number of 40 Points.

Code	**0: totally disagree, ..., 4: totally agree**
Interfaces remained stable with respect to the design document handed in in step 2.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The code is cleanly designed (e.g. uses appropriate grouping of features and classes, consistent naming conventions, meaningful class and feature comments).	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The code satisfies the functional requirements of the SRS with top priority.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The code satisfies the non-functional requirements of the SRS where applicable.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The classes include relevant contracts.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The code is extensible, reusable, and efficient.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The overall impression of the code is excellent.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
Extended design document	
The new parts of the design document describe any major changes that needed to be done.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The reflection on the initial design uses a good line of argumentation.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
Presentation	
The presentation covered the required topics.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]

5. Test report

Deadline: Sunday, 23.05.10
Time: 7 days

The implementation of your project submitted in step 4 will now be delivered to the project team that developed the test suite based on your design document, code base, and SRS. In turn, you will receive the code of the team that you wrote the test suite for.

Your task:

Run the test suite on the received code base. Then write a short report summarizing the results. Identify the test cases that were successful/failing and give an overview of the percentages of failing/successful test cases. Write a short paragraph on how you estimate the quality of the provided code and the quality of your test suite, highlight which parts of the project are reliable, and which parts are error prone, also highlight the parts which are difficult to test, and explain why. If you cannot run all test cases that you initially designed, describe why. On the example project page, there is a template for the test report http://soft-arch-project.origo.ethz.ch/wiki/test_report.

When you find bugs in the project, submit bug reports using the "Create Issue" facility in Origo. Note that while all documents so far are part of your own project, the issues now must be submitted on the Issue page of the **tested project**. From your test report provide links to the submitted issues.

Resources:

- Test report template (http://soft-arch-project.origo.ethz.ch/wiki/test_report)
- Tested project's source code (*https://svn.origo.ethz.ch/[**other-project**]/src*)
- Your test plan document from step 3 (*http://[your-project-name].origo.ethz.ch/wiki/test_plan*)
- Your test suite from step 3 (*https://svn.origo.ethz.ch/[your-project-name]/test*)

Deliverables:

- Test report available at or linked from *http://[your-project-name].origo.ethz.ch/wiki/test_report*.
- A set of bug reports on the tested project's Issues page *http://[**other-project**].origo.ethz.ch/issues*, linked from your test report.

Grading criteria:

We will use the following grading scheme for the test report with a maximum number of 12 Points.

Test results	**0: totally disagree, ..., 4: totally agree**
The test report gives a correct overview of the project quality from the testing point of view.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
Parts in the project that are reliable, error prone or difficult to test are clearly highlighted.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
Issued bug reports provide necessary information to understand and reproduce bugs.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]

6. Revision and release

Deadline: Friday, 04.06.10
Presentation: Monday, 31.05.10
Time: 12 days

In this project step, you will just have received the test report and the reported issues from your test engineers. Based on this, you will prepare your project for release.

Your task:

Go through the reported issues and for every issue decide whether it needs a fix or not. Comment every issue by classifying it: (1) it was a bug and was fixed; (2) it was a bug, but could not be corrected (why?); (3) you were not able to reproduce the problem; (4) it was not a bug. Fix the bugs that you feel should be fixed. Decide on the parts of your project that should be released (you may decide to exclude unstable parts) and finally, prepare a release for your project and put it on your Origo project's download page (see http://soft-arch-project.origo.ethz.ch/download).

In the final presentation you should shortly discuss how well your test plan from task 5 performed. Explain if you had any problems in running the test plan and why. Then, describe how you worked on the project, how you distributed the work and handled communication in the group. As a conclusion of the course, give your personal impression of the project: are you happy with the result? would you do some part differently now? The presentation should be between 10 and 15 minutes long (= 8-12 slides).

Resources:

- Testing team's test plan (*http://[**testing-team**].origo.ethz.ch/wiki/test_plan*)
- Testing team's test report (*http://[**testing-team**].origo.ethz.ch/wiki/test_report*)
- Testing team's test suite (*https://svn.origo.ethz.ch/[**testing-team**]/test*)
- Issues submitted by testing team (*http://[your-project-name].origo.ethz.ch/issues*)
- Source code of step 4 (*https://svn.origo.ethz.ch/[your-project-name]/src*)

Deliverables:

- Commented issues on *http://[your-project-name].origo.ethz.ch/issues*.
- Release on origo available at *http://[your-project-name].origo.ethz.ch/download*.
- Slides of your presentation on SVN available at *https://svn.origo.ethz.ch/[your-project-name]/presentation*.

Grading criteria:

We will use the following grading scheme for the final release with a maximum number of 20 Points.

Bugs and final release	**0: totally disagree, ..., 4: totally agree**
The choices of bugs to fix and bugs to leave open are reasonable and justified.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The release includes a reasonable amount of functionality.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The released code needs no improvements to be usable by others.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
The overall quality of the release is excellent.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]
Presentation	
The presentation covered the required topics.	0[ ] 1[ ] 2[ ] 3[ ] 4[ ]