CONCORDIA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

COMP 354: INTRODUCTION TO SOFTWARE ENGINEERING: SECTION BB
SUMMER 2019

`ETERNITY`: A SCIENTIFIC PROJECT IN THE 'SMALL'

MOTIVATION

Instead of learning big methodologies [...] focus on practices. [...] You can become an expert in a practice without being an expert in a complete methodology.
— Ivar Jacobson

The software industry has grown to become the largest industry. However, it had humble beginnings.

It is likely that **calculator** was the first 'educational' tool that many prospective economists, engineers, and scientists, were exposed to during their time in high school, and was a tool that could do something that they could not do, sometimes at all and other times at the same speed. It should therefore come as no surprise that many operating systems provide native support for a calculator.

A question of interest for a software engineer is how a typical scientific calculator does what it is supposed to do.

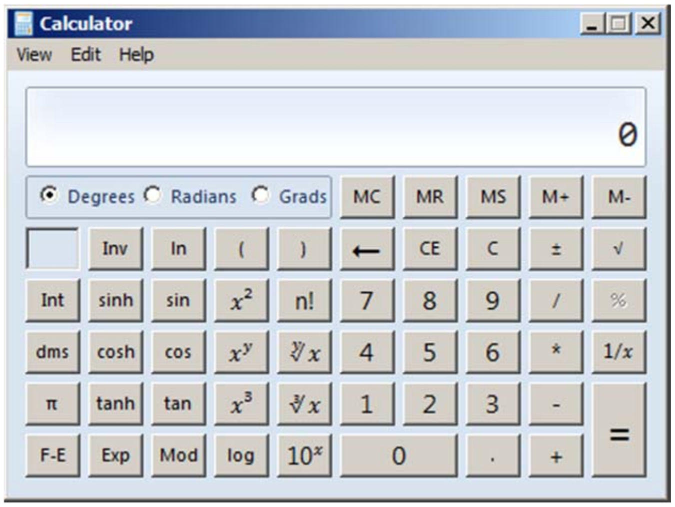
The **question may be simple**, but the **answer is not (as simple)**. The answer is also crucial towards providing an insight as well as an in-depth understanding of how more challenging numerical or symbolic calculations are performed by the state-of-the-art scientific computing software.

DESCRIPTION

This is an elementary **scientific software engineering project**, with an injection of certain elements of **agile methodologies**. It involves both individual work as well as communal work.

A **transcendental function** is a function that does not satisfy any single-variable polynomial equation whose coefficients are themselves roots of polynomials. For example, sin(x) is a transcendental function, but √x is not.

Let N be the **size** of a project team. The purpose is to compute, **from scratch**, **N (mostly) transcendental functions** found on a typical scientific calculator[1], such as the one shown below:


Calculator
View Edit Help
0
Degrees Radians Grads
MC MR MS M+ M-
Inv ln ( ) ← CE C ± √
Int sinh sin x^2 n! 7 8 9 / %
dms cosh cos x^y $\sqrt[y]{x}$ 4 5 6 * 1/x
π tanh tan x^3 $\sqrt[3]{x}$ 1 2 3 -
F-E Exp Mod log 10^x 0 . + =

The permitted set of choices is:

1. sin(x) XOR cos(x)
2. 10^x XOR π^x
3. $\log_e(x) = \ln(x)$ XOR $\log_{10}(x)$
4. e^x XOR a^x
5. √x XOR σ (Standard Deviation)
6. sinh(x) XOR cosh(x) XOR tanh(x)
7. x^y

XOR means "exclusive or". The variables x and y are real numbers, unless otherwise stated.

Any **subordinate functions** developed by the team could be added to the above set.

[1] To be precise, the transcendental functions on a calculator are known as **elementary functions** to distinguish them from special functions.

CONSTRAINTS

Domain	The transcendental functions chosen must be **unique across a team**. (In other words, each team member must select his or her own transcendental function.)
Interviews	There must be **N interviews with different interviewees** conducted by each team. The interviewees must be **potential users** of the **`ETERNITY`**.
Personas	There must be **N personas** developed by each team.
Use Cases	The use cases must be decided by a team entirely by itself, and must be **common** across a team.
Design	This is for Iteration II. It is important for the design to be as **independent** as possible of the technologies underlying the implementation.
Implementation	The implementation technology, that is, the programming language, could be any high-level (object-oriented) programming language **suitable for scientific computing**, such as Java or Python. The term "scratch" means that apart from the functions related to input, output, and arithmetic, an **`ETERNITY`** implementation should **not** make use of any built-in functions provided by a programming language. The user interface could be either **textual or graphical**. The environment could be **distributed** (such as, the Web) or **non-distributed** (such as, native to the operating system). The **`ETERNITY`** implementation should aim to be **correct, efficient, maintainable, robust, and usable**.
Testing	A **transcendental number** is a real number that is not the solution of any single-variable polynomial equation whose coefficients are all integers. For example, π is a transcendental number. As a test case, each transcendental function in **`ETERNITY`** must be able to (1) provide an output for an input of 0, and (2) compute the value of at least one transcendental number, with a reasonable accuracy (that is, **small relative error**). For example, $\sin(\pi) = 0$, $e^0 = 1$, and $\sqrt{2} = 1.41421$ …

Glossary	The teams must develop a glossary that should be maintained throughout the duration of the project. It is preferable to develop the glossary alongside (specifically, in **parallel** with) other activities. The glossary should contain definitions of those terms that are likely **not commonly-known** to most stakeholders of the project.
Team Interaction	It is important that each team **meet face-to-face regularly** (at least once every week), and document meeting minutes. The teams must be aware of **collaboration patterns** and **use social software** for project-related concerns. For example, a **Wiki hosting service** is such social software. Let S(X) be such social software for Team X. Then, X must make available the **roles and responsibilities** of each of its members, pertaining to each deliverable, publicly on S(X), soon after the teams are announced. The responsibilities must include **asymmetric peer review of source code**. (If A and B are two members of a team, and A reviews B's source code, then B should not review A's source code.)

ITERATION I

This has two deliverables.

DELIVERABLE 1

This involves providing an initial set of personas and use cases, both informed by interviews, an implementation of those use cases, and testing the implementation against those use cases.

It is expected that the personas and use cases are expressed in some proper format and the source code for the entire team follows some publicly-accepted, commonly-agreed-upon, programming style.

It should be noted that there is **decision-making** throughout development. For example, a given function can usually be computed in more than one way. It is acceptable to reduce the set of choices to not more than three. In the end, this leads to selecting **one** option from a given set of choices, knowing that each choice has its own advantages and disadvantages.

DELIVERABLE 2

This involves a presentation of how the development is organized, and a demonstration of an initial version (prototype) of **`ETERNITY`** in the class.

SUBMISSION

This involves an archived submission of (1) documentation (collaboration patterns followed; selected functions; initial list of potential personas; list of interview questions and answers; initial set of use cases; an outline of the strategy, including algorithm(s) expressed as pseudocode or otherwise in some 'standard' form and data structures, used for implementation, **inclusions and exclusions** for Iteration I (to be addressed in Iteration II), and technical reasons for making decisions (such as, those for **inclusions and exclusions**); source code review results (if any); test results; and instructions on how to run the program), (2) source code, (3) data files (if any), and (4) a copy of the presentation slides. The work by others should be cited and referenced accordingly.

ITERATION II

This has two deliverables.

DELIVERABLE 3

This involves providing the final set of personas and use cases, both informed by interviews, a design based on those use cases, an implementation of that design, and testing the implementation against those use cases.

It is expected that the personas and use cases are expressed in some proper format and the source code for the entire team follows some publicly-accepted, commonly-agreed-upon, programming style.

It should be noted that there is **decision-making** throughout development. For example, a given function can be computed in **more than one way**. It is acceptable to reduce the set of choices to not more than three. In the end, this leads to selecting **one** option from a given set of choices, knowing that each choice has its own **advantages and disadvantages**.

It incorporates revision (if necessary) of the software artifacts from Iteration I. For example, the final set of use cases will include the initial set of use cases, with changes if necessary, based on the feedback from Deliverable 1 and Deliverable 2.

This is regarding the use of the **After-Scenario Questionnaire (ASQ)**[2] for evaluating **ETERNITY**. ASQ is based on the publication entitled **IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use**[3].

- Each member of every team must prepare **one** task for his or her respective function. This will lead to N tasks per project team, where N is the size of the team.

- Each team must contact one of the Teaching Assistants, no later than **Monday, July 15, 2019**, to get permission (confirm) that she will complete the ASQ evaluation. (The onus of making such an arrangement, and having ASQ evaluation completed by the stipulated date, is on the team, not on the Teaching Assistants.) The Teaching Assistants will randomly select **three** out of the N tasks, evaluate them using ASQ, and report the results in confidence. The teams should make the Teaching Assistants aware of their tasks (say, by e-mailing the tasks) no later than **Monday, July 29, 2019**, so that the Teaching Assistants have adequate time to decide the tasks to select.

[2] URL: http://garyperlman.com/quest/quest.cgi?form=ASQ .
[3] URL: http://www.tandfonline.com/doi/abs/10.1080/10447319509526110 .

New Constraints

The new constraints are about bringing an **unanticipated change** in the process, much like agile development methodologies.

New Constraint 1

Dan's part

There should be **software architecture description** for the **macro-architectural design**. There should be a **CRC model** for the **micro-architectural design**. The design at any level, including the user interface design, should carefully select **software patterns** and make use of them wherever appropriate.

New Constraint 2

Each member of every team must demonstrate the use of a **debugger**. For example, `gdb` is a comprehensive, general-purpose, debugger, and `pdb` is a debugger for Python. This could be done by including a **screenshot** of use.

New Constraint 3

Every team must conform to the **same, recognized, programming style** for the programming language underlying its **`ETERNITY`**. For example, **Python Enhancement Proposal 8 (PEP 8): Style Guide for Python Code**[4] is the 'standard' programming style for Python.

DELIVERABLE 4

This involves a demonstration of a final version of **`ETERNITY`** in the class, and a poster presentation of retrospectives from Iteration I and of lessons learned (if any) from design. The presenters of this deliverable should be different from those in Deliverable 2 so as to give others an opportunity to present.

SUBMISSION

This involves an archived submission of (1) documentation (collaboration patterns followed; selected functions; final list of potential personas; final set of use cases; an outline of the strategy, including specifics of software design (macro-architecture design, micro-architecture design, and user interface design), algorithm(s) expressed as pseudocode or otherwise in some 'standard' form and data structures, used for implementation, and technical reasons for making decisions; source code review results; test results; usability evaluation results; and instructions on how to run the program), (2) source code, (3) data files (if any), and (4) an electronic copy of the poster. The work by others should be cited and referenced accordingly.

[4] URL: https://www.python.org/dev/peps/pep-0008/ .