INTRODUCTION TO EIFFEL

TIE-20306 Principles of Programming Languages

Group 25: Likai Ren, Jundong Wang
1 DESIGN GOALS

To make Eiffel a decent programming language to help specify, design, implement and modify quality software, there are some factors that Eiffel follows in the language design. Reusability, extendibility and reliability are the fundamental factors, as well as other also significant factors such as efficiency, openness and portability.

Reusability is the ability to create some components which can be used in numerous different applications, and make it possible to write less code. In the Eiffel, there are many predefined libraries that can be reused, and it is supported to produce new libraries by programmers.

Extendibility is the ability to produce software that can be modified easily. Since it is normally so difficult to modify a large software system, the extendibility of Eiffel is important to solve this problem.

Reliability is the ability to produce reliable – robust and bug-free software. Eiffel implements various techniques such as disciplined exception handling, assertions, and garbage collection to reach the reliability.

Eiffel enables programmers to implement efficiency compilers to make the systems developed by Professional Eiffel effective enough, under speed and space conditions.

Openness makes Eiffel software easy to couple with non-Eiffel programs.

Portability is the ability to produce cross-platform software by the platform-independent language definition of Eiffel, which means the same semantics work on many platforms.

2 DESIGN PRINCIPLES

Some important programming principles are introduced by Eiffel, including design by contract, the uniform-access principle, the open/closed principle, the single-choice principle, command-query separation, and information hiding.

Design by contract means that software components in a system cooperate based on precisely defined contracts which are represented by preconditions, postconditions, and class invariant. This principle ensures program correctness but not at the expense of sacrificing efficiency. This is the most important contribution for Eiffel to the software engineering.

The uniform access principle states “all services offered by a module should be available through a uniform notation, which does not betray whether they are implemented through storage or through computation”, which means that it should be the same notation between working with an attribute, precomputed property, and method/query.

The open/closed principle states “software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification”, which means that such a software entity only allows to be extended but without modifying its source code.
The single-choice principle known as “don’t repeat yourself” principle, states that “Every piece of knowledge must have a single, unambiguous, authoritative representation within a system”, aiming at reducing repetition of information of all kinds, especially useful in multi-tier architectures.

Command-query separation divides methods into two categories: command that performs an action, and a query that returns value to the caller.

Information Hiding constrains the accessibility of certain parts of a software component to its clients by programming language features or explicit exporting policies.

3 MECHANISMS AND FEATURES

3.1 PARADIGM
Eiffel is a multi-paradigm programming language which supports many programming paradigms, such as object-oriented programming, generic programming and event-driven programming.

As a typical representative of object-oriented programming (OOP) languages including such as Smalltalk, C#, and Java, apparently, Eiffel has the most prominent programming paradigm object-oriented programming, in which structured collections of abstract data type “class” constructs the software system to reach reusability, extendibility and reliability.

Generic programming is other programming paradigm of Eiffel. Types can be specified and instantiated later as parameters. This paradigm as well as the inheritance feature of the class, help to ensure reusability and extendibility of the software.

Event-driven programming is also supported in Eiffel. In this paradigm, events such as user actions drives the flow of the program, and it is the most significant paradigm for Graphical User Interfaces (GUI).

3.2 OVERALL STRUCTURE
An Eiffel program consists of classes, above which cluster is defined to hold a group of classes or subclusters. A class includes invariants, features (similar to “attributes” or “methods” in other object-oriented programming languages), and other properties. There should be a “root” class with “root procedure” as its creation procedures.

There are five basic executable instructions in Eiffel, including assignment, object creation, routine call, condition, and iteration.

3.3 “HELLO WORLD” FROM EIFFEL

class
HELLO_WORLD
create
make
feature
make
In this snippet, the class name is “HELLO_WORLD”; the constructor “make” invokes the “print” system library routine to output “Hello world!”

### 3.4 MEMORY MANAGEMENT
Eiffel possesses garbage collection to hinder program errors from negatively affecting software’s performance and efficiency while from manual memory management, thus, no other memory leakage always takes place. We may take into consideration unleashing memory resources in time, for instance, closure of a document rather than delaying the closure till the object may be targeted to collect as garbage. On some occasions, the garbage collector that we inevitably desire to set up, however, these special cases are not commonly seen.

### 3.5 TYPING & PARAMETER PASSING
According to the explicit definition of static typing—the result of the simplified object-oriented computation modelling, provided that one lays aside the particulars of one object-oriented programming language, and necessarily create pragmatic software which may be supplementary to the fundamental model, merely one type of event takes place on the process of the OOP system’s execution—routine call. In commonly written form below, by virtue of the syntax of Eiffel, that is

\[ y.g(arg) \]

: implement on the object tied to \( y \) that operation \( g \), utilizing that argument \( arg \), having the knowledge that at times \( arg \) represent some arguments. During run time, it may be what the systems execute: calling characteristics from objects, delivering arguments as entails, which depends on the canonical scheme.

Eiffel typically passes most variables by reference. Generally speaking, in Eiffel the general “value” possessed by the entity may be a reference to an Eiffel object. These “values” passed offer reference passing semantics. Nevertheless, we could not definitely pass a reference to the entity, but simply to that object which it targets, which may be different from languages C++, in which not only one pointer may be passed, but also one pointer to one pointer, etc. Eiffel holds distinctive types considered as expanded. One expanded type may be passed on through value. For examples, INTEGER, and even predeclare any class as expanded.

### 3.6 DESIGN BY CONTRACT & ASSERTION
While the failure of an assertion occurs, Eiffel may throw an exception. That exception may be captured by a function, otherwise returned to that caller. Provided that the exception is returned directly to a top function leading to the failure of the program, a debugger may be referred to by the implementations while others only print out the stack trace. Ada will find out the path to the “runtime” checks. Nevertheless, Ada is quite confined to checks on language constructs. Eiffel affords a practical compromise.
So far, all Eiffel environments contain a filter to produce the documentation spontaneously. By definitely showing the contract among clients and those server classes, assertions may benefit inhibiting misleading among programmers to in order to facilitate integration. These laws make sure that the child may be precisely replaced by the parent. Errors are always captured when taking place to avoid manual faults and carelessness, while the type of exception mechanism does not exist in C++ programming language.

Assertions may be instrumental in Eiffel methodology for constructing and developing credible OOP software. They are always ready to make some definite presumptions that developers depend on while writing software compositions considered to comply with paradigm. Utilizing assertions may be accessible to illustrate clearly the contract ‘s terms that may regulate the connectivity from a routine to those callers. The precondition may constrain callers, in contrast, postcondition may constrain the routine.

3.7 INHERITANCE MODEL
While a new class is announced as a successor to the last predefined one, it owns by inheritance the whole features from the parent’s class as well as their affiliated formal characters. The descended features may not be declared again in new class; however, new features are allowed to be enriched. Inherited features apart from new ones may underlie the class and could be delivered to relevant classes decided by birth, where software may be constructed effectively not under the sole setting but by virtue of progressive accumulation.

In terms of the syntax indicating above, inheritance which is supplied by Eiffel may be multiple, where a class might descend from diverse classes when necessary. The mere restriction may come from the acyclic inheritance graph.

4 REFERENCE


https://www.wikiwand.com/en/Programming_paradigm

https://www.eiffel.org/doc/eiffel/Eiffel

http://se.inf.ethz.ch/courses/2013b_fall/eprog/additional_materials/eiffel_the_essentials.pdf