2017

SOFTWARE DEVELOPMENT

ORLANDO - FLORIDA

CONTENT

1. Research about the problem; one (1) page
2. Description of the team’s project, including: the problem, solution for the problem, and an explanation of the project’s educational and social value; up to two (2) pages
3. Documentation of the use of a software development process, including the following:
a. Project requirements; one (1) page
b. High-level software design; one (1) page
c. Testing, including code output and desired results; pages as needed
d. End-user product documentation; pages as needed
4. Team’s self-evaluation (of its work) and the project’s future prospects; one (1) page
5. List of references used for the project

RESEARCH

Heading away from the fact that science and technology shape our future, we want to make a contribution to education. According to the survey* results, the most popular subject of late years is the computer science. Considering the mass of people interested in computer science enlarges directly proportional to the popularity, more problems come up and wait to be solved during the learning process. There are some aspects are worth to take a look at, in order to specify the problem that we are going to deal with.

First of all, the requirements for a code to be well written should be listed. Readability, modularity, expressivity and efficiency are the main concerns about a good program. Different techniques for readability exist, whereas expressivity and efficiency are more complicated and depend mainly on the programmer. Therefore modularity builds a bridge in between. A system's components may be separated via self-contained sequences of actions to be performed, as known as algorithms. In an attempt to achieve the most efficient written code, a programmer must acknowledge the working principle of algorithms. So that one can calculate the estimated run-time and necessary memory. Only then one gain the ability to optimize a code.

A nonignorable method for understanding how an algorithm works is called abstraction*, which is a technique for arranging complexity of computer systems. When it comes to the education process of programming, a big majority of students face problems, especially about abstraction. For instance, a few members of our chapter including us are preparing for the USA Computing Olympiad. In a given interview about how hard is it to learn and study programming, it is mentioned that programming class has a high rate of daunting mainly because of the inability of the comprehension the concept of abstraction.

As a conclusion, we decide to find an easier way of teaching in order to smooth over the adaptation period for abstract thinking.

1)http://www.telegraph.co.uk/education/educationpicturegalleries/10643255/Student-life-top-ten-most-popular-subjects.html
2)https://en.wikipedia.org/wiki/Abstraction_(software_engineering)

DESCRIPTION

As a result, we decide to improve the way of teaching intermediate level programming by bringing in more visualization in contrast to the abstraction.

what is our problem, why is that a problem, how we try to solve it, why that way, what new windows had it brought out, how important is our problem, why so important, why we decided to work on that
suffer from the abstract flow
visual learning
interactive (to answer MY questions, I decide the graph, I ask statements to an application, I get my OWN answer in a visual way)
what other innovations were made
why they were not enough

REQUIREMENTS

System Properties

Operating System

Windows 7 and later are supported, older operating systems are not supported (and do not work).
Both x86 and amd64 (x64) binaries are provided for Windows. Please note, the ARM version of Windows is not supported for now.

Linux is supported following are guaranteed to work:
Ubuntu 12.04 and later
Fedora 21
Debian 8
-
Hardware

Windows
An Intel Pentium 4 processor or later that's SSE2 capable
512 MB of RAM

Linux
An Intel Pentium 4 processor or later that's SSE2 capable
-

REQUIREMENTS (PRD)

system properties: operating system, hardware
specifications : A software requirements specification (SRS) is a description of a software system to be developed. It lays out functional :
A functional specification a document that specifies the functions that a system or component must perform.A functional requirement in a functional specification might state as follows:

When the user clicks the OK button, the dialog is closed and the focus is returned to the main window in the state it was in before this dialog was displayed.
The plan for implementing functional requirements is detailed in the system design.

and non-functional requirements:
In systems engineering and requirements engineering, a non-functional requirement (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors.The plan for implementing non-functional requirements is detailed in the system architecture, because they are usually Architecturally Significant Requirements

and may include a set of use cases that describe user interactions that the software must provide.

Purpose and scope, from both a technical and business perspective
In project management, the term scope has two distinct uses: Project Scope and Product Scope.
Scope involves getting information required to start a project, and the features the product would have that would meet its stakeholders requirements.
  Project Scope: "The work that needs to be accomplished to deliver a product, service, or result with the specified features and functions."
  Product Scope: "The features and functions that characterize a product, service, or result."
Notice that Project Scope is more work-oriented (the hows), while Product Scope is more oriented toward functional requirements (the whats).

Stakeholder identification
Market assessment and target demographics
Product overview and use cases
Requirements, including
- functional requirements (e.g. what a product should do)
- usability requirements
- technical requirements (e.g. security, network, platform, integration, client)
- environmental requirements
- support requirements
- interaction requirements (e.g. how the product should work with other systems)

Assumptions
Constraints
Dependencies