# Rewriting the original program:

## Introduction

Importance of naming, earlier finds (Butler, et al. 2009)

|  |  |  |
| --- | --- | --- |
| Name | Description | Example of flawed identifier(s) |
| Capitalization Anomaly | Identifiers should be appropriately capitalized | HTMLEditorKit, pagecounter |
| Consecutive Underscores | Consecutive underscores should not be used in identifier names | foo\_\_bar |
| Dictionary Words | Identifier names should be composed of words found in the dictionary and abbreviations, and acronyms, that are more commonly used than the unabbreviated form. | strlen |
| Excessive Words | Identifier names should be composed of no more than four words or abbreviations | floatToRawIntBits() |
| Enumeration Identifier Declaration Order | Unless there are compelling and obvious reasons otherwise, enumeration constants should be declared in alphabetical order | enum Card {ACE, EIGHT, FIVE, FOUR, JACK, KING ...} |
| External Underscores | Identifiers should not have either leading or trailing underscores. | \_foo\_ |
| Identifier Encoding | Type information should not be encoded in identi- fier names using Hungarian notation or similar | int iCount; |
| Long Identifier Name | Long identifier names should be avoided where possible | getPolicyQualifiersRe-jected |
| Naming Convention Anomaly | Identifiers should not consist of non-standard mixes of upper and lower case characters. | FOO\_bar |
| Number of Words | Identifiers should be composed of between two and four words. | ArrayOutOfBoundsExcep-tion, name |
| Numeric Identifier Name | Identifiers should not be composed entirely of numeric words or numbers | FORTY\_TWO |
| Short Identifier Name | Identifiers should not consist of fewer than eight characters, with the exception of: c, d, e, g, i, in, inOut, j, k, m, n, o, out, t, x, y, z | name |

Butler et al and comments on finds

Further definitions and conventions

|  |  |  |
| --- | --- | --- |
| **Identifier** | **Case** | **Example** |
| Class | Pascal | **AppDomain** |
| Enumeration type | Pascal | **ErrorLevel** |
| Enumeration values | Pascal | **FatalError** |
| Event | Pascal | **ValueChanged** |
| Exception class | Pascal | **WebException** |
| Read-only static field | Pascal | **RedValue** |
| Interface | Pascal | **IDisposable** |
| Method | Pascal | **ToString** |
| Namespace | Pascal | **System.Drawing** |
| Parameter | Camel | **typeName** |
| Property | Pascal | **BackColor** |

Table some camel/pascal styles from (Cwalina og Abrams 2005)

What was compared?

* Pascal/camel/underscore/hypen
* Expansion of acronyms/keeping acronyms
* Large scope /small scope length
* Fine granule modulation / larger modules (less overhead)

What problems were solved, through comparison?

* Found bug

Why was this important?

* Established practice to be upheld by later contributors
  + Rewrite to C
  + Establish testing
  + Expansions

What was the specific hypothesis?

* Constant added overhead, no polynomial increase

How did you test this hypothesis?

* Tic toc

What insights does comparison reveal? Why specific approach useful?

## Method

1: Identify algorithms

2: Explain Implementations

* Editing names
* Modularizing into classes and functions
* Editing comments
  + Example, the battery has been charged for the last eight hours, but not this one
    - May lead reviser to believe that there is a bug

## Results / Discussion