The Art of Design and Development

Ravishankar Rajagopalan

Why do we need it ?

- **Programmers spend 49% of their time debugging***
- Total estimated cost of debugging is **$312 billion**
- **Need to significantly improve productivity!**

**Source CVP Surveys*

Why do we need it ?

- Spent hour(s)/day(s) to find a bug which could have been found in minutes
- Maintenance nightmare
  - Realized later that we used overly complex data structures/coded wrong algorithms
  - Nobody can understand the program now (at times, not even the author)
- Need to make the program run faster, use less memory
- Tested a program but missed some obvious use cases
- Program works fine in our system, cranks in production
- Struggled to port a program
- Realized later we could have automated many more things
- Need Refactoring
- ..

Where do we start from ?

- Reliability
  - Does not mean system is always working perfectly or giving desired results
  - Fault within tolerance limits
- Availability
  - Up-time or down-time
- Serviceability
  - Easy to maintain/upgrade/deal with production issues

Everything we do and talk will be around this!

Slowness can be an acceptable trait, but failure and data loss are almost never acceptable

Principles

- Simplicity
  - Keep programs small and manageable
- Clarity
  - Easy to read and understand (esp for people)
- Generality
  - Work under a wide range of situations
  - Take this with a pinch of salt
    - There are situations when development time could be abnormally high esp. When trying to generalize when not needed
- Consistency
- Maintainability
- Automation
  - Automate as much as possible
  - Makes it easy to test too

Helpful for

- Professionals/Engineers
- Managers
- Architects
- Anyone who’s willing to be a student in IT/software design & development

Learn by Example

- May use C/Java/Go/Python other programs to demonstrate, but ideas are much language agnostic
- Examples are almost entirely production code
- Some of us may know many or all of these things & few of us apply this unconsciously
  - Always good to learn from professionals who know the craft

Naming

- Use descriptive names for globals and short names for local variables

```
int numberOfPosts = 4;

for (elementIndex = 0; elementIndex < numberOfPosts; elementIndex++)

    elementArray[elementIndex] = elementIndex
```

```
int nposts = 4;

for (i = 0; i < nposts; i++)

    el[i] = i;
```

- Naming conventions
  - May indicate what the variable does/type
    - `stptr` or `nodep`
    - `zeta_str, alt1_int64`

..Naming

- Globals
  - Well differentiated from the local (and possibly) package variable names to avoid mixups
  - More descriptive
    - Use `elementIndex` instead of `elx`
  - Unique package/other name prefixed if it’s going to be multiple packages merged together
    - `int CralSecNumQItems; or __cralSecNumQItems//depending on language`
  - Constants can usually be all caps
    - `CRALINA_SEC_MAX_Q_LENGTH = 100 //e.g. Cralina’s security package, max Q length`
- Camel case or _ or something else?
  - Better to use convention used by language or package developer/organization
  - Don’t mixup - *use ONE convention in all of your package in a given language*
    - If all existing code is all using _, use _ even if language convention is otherwise, **consistency is more important than convention**

..Naming

- Relevant and non-repetitive
    - Well differentiated from the local (and possibly) package variable names to avoid mixups
    - More descriptive

```
class ItemQueue {
    int numItemsInQ, queueCapacity, frontOfQueue
    public int numberOfItemsInQueue() {...}
}
```

```
class ItemQueue {
    int nitems, capacity, front
    public int getNumItems() {...}
}
```

..Naming

- Self explanatory
  - Use intuitive active verbs

```
if(checkDigit(c)) ..
```

```
if(isDigit(c))..
```

Bad Naming Real life Samples

```
#define FALSE 1
#define TRUE 0

if ((ch == getchar()) == EOF) {
    not_eof = FALSE;
}
```

```
#define FALSE 0
#define TRUE 1

if ((ch == getchar()) == EOF) {
    eof = TRUE;
}
```

Bad Naming Real life Samples

```
if ((falloc(SMRHSHSCRTCH, SJFEXT| 0644, MAXRODDHSH)) < 0)        //God knows ?
...
```