

# Design and Implementation

The Answer to Life, the Universe and Everything

Hans Verkuil  
Cisco Systems Norway

# Overview

- Design and implementation guidelines.
- Simple techniques to keep your code nice and short.
- General guidelines to help you stay sane.

# Disclaimer

- These are guidelines only, for every guideline I can think of exceptions where it doesn't apply. In other words, use common sense.
- This presentation reflects my opinions, not those of my employer.
- I am not responsible for any adverse effects this presentation might have on your career. I am, of course, responsible for any positive effects this presentation might have.

# Design and Implementation Guidelines

# Do As Little As Possible!

# Design Guidelines

- Initial design should be fairly high-level. So don't start designing function prototypes, stick to the global picture.
- Identify which parts are 'risky' because you do not know how to implement it, or do not know how long it will take to implement it, or do not know if it even can be done.
- Highly dependent on experience and capabilities of the S/W engineer.
- Start with the high-risk parts. Just research and implement enough to be able to test these parts of the code.
- Try to plan the time you'll need to research these high-risk areas.

# Implementation Guidelines

- Evolutionary prototyping.
- Coding == learning.
- Design → coding → learning → design → ...
- Avoid too many layers: that is hard to debug and understand.
- Avoid too few layers: a certain amount of abstraction makes the code much more understandable.
- Lean and mean: only code what you need, not what you might need. But keep the door open for future developments.
- It is your responsibility to warn your project lead as soon as you know you are not going to make the planning!

# Code Classification

- Absent code: that's where you earn your pay.
- Working code: that's where someone earned his pay.
- Buggy code: that's where someone didn't earn his pay, but you will.
- **Dead code**: that's when someone should be fired and you are really going to earn your pay.



Keep It Simple,  
Keep It Short

*Increasingly, people seem to misinterpret complexity as sophistication, which is baffling: the incomprehensible should cause suspicion rather than admiration.*

Niklaus Wirth

# Keep It Simple, Keep It Short

- Keeping it simple is very, very hard.
- Keeping it short is much easier and is the first step to simplicity.
- Short code is easier to understand and therefore easier to maintain.
- Short code highlights bad code which might be a candidate for a rewrite.
- Removal of non-essential code does not help a bad design. But it *does* make a bad design more obvious.

# Example 1: Keep It Short

```
/**
@see DVBT_DEMOD_FP_IS_SIGNAL_LOCKED
*/
int
rtl2832_IsSignalLocked(
    DVBT_DEMOD_MODULE *pDemod,
    int *pAnswer
)
{
    unsigned long FsmStage;

    // Get FSM stage from FSM_STAGE.
    if(pDemod->GetRegBitsWithPage(pDemod, DVBT_FSM_STAGE, &FsmStage) != FUNCTION_SUCCESS)
        goto error_status_get_registers;

    // Determine answer according to FSM stage.
    if(FsmStage == 11)
        *pAnswer = YES;
    else
        *pAnswer = NO;

    return FUNCTION_SUCCESS;

error_status_get_registers:
    return FUNCTION_ERROR;
}
```

# Example 1: Rewritten

Old code: 32 lines. New code: 13 lines.

```
/**
@see DVBT_DEMOD_FP_IS_SIGNAL_LOCKED
*/
int rtl2832_IsSignalLocked(DVBT_DEMOD_MODULE *pDemod, int *pAnswer)
{
    unsigned long FsmStage;

    if (pDemod->GetRegBitsWithPage(pDemod, DVBT_FSM_STAGE, &FsmStage) != FUNCTION_SUCCESS)
        return FUNCTION_ERROR;

    *pAnswer = (FsmStage == 11) ? YES : NO;
    return FUNCTION_SUCCESS;
}
```

# Example 2: Keep It Short

```
/* 19 lines of comment removed */
AiUInt8 UnivWriteRegister(AiUInt32 RegBaseAddr,
                          AiUInt16 Register, AiUInt32 regValue)
{
    AiUInt32  *addr_value, regvalue;

    regvalue   = (BSWAP32(regValue));

    addr_value = (AiUInt32*) (Register + RegBaseAddr);

#ifdef AVI_REGISTER_IO_TEST
    printf("____UnivWriteRegister: Address:%08x / Data=%08x(BE), %08x(LE)\r\n", (int)addr_value,
          (int)regValue, (int)regvalue);
#endif

    *addr_value = regvalue;

    return(TRUE);
} /* end: UnivWriteRegister */
```

# Example 2: Rewritten


Old code: 36 lines, new code 11 lines

```
/*! Write the specified register value to the specified
 * register offset in the Universe memory space.
 *
 * \param RegBaseAddr Register base address in Universe memory space
 * \param Register Offset from the register base address
 * \param regValue The value to be written
 */
void UnivWriteRegister(uint32_t RegBaseAddr, uint16_t Register, uint32_t regValue)
{
    *(uint32_t *) (RegBaseAddr + Register) = aviCpuToPci(regValue);
}
```

# Spacing

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her

sister was reading, but it had no pictures or conversations in it, “and what is the use of a book,” thought Alice “without pictures or conversation?”




Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, “and what is the use of a book,” thought Alice “without pictures or conversation?”



# Example 3: Spacing

```
bool RTSPMsg::findCSeq(char *line){
    char *findCSeq = strstr(line,"cseq:");
    int seqNum = 0;
    if(findCSeq && ( sscanf(findCSeq,"cseq: %d",&seqNum) == 1)){
        aCSeq = seqNum;
        return true;
    }else{
        return false;
    }
}
```



```
bool RTSPMsg::findCSeq(const char *line)
{
    const char *findCSeq = strstr(line, "cseq:");
    int seqNum = 0;

    if (findCSeq && sscanf(findCSeq, "cseq: %d", &seqNum) == 1) {
        aCSeq = seqNum;
        return true;
    }
    return false;
}
```

# Example 4: Ridiculous Comments

```


/*****
/*
/*  Module : API_SYS          Submodule : API_SYS_RESET
/*
/*  Author : XXXXX XXXX      Project   : FOOBAR
/*
/*  Source : C                Tools    : PC/AT; Norton Editor;
/*                                  CYGNUS, GNU-C, As, and LD
/*                                  IDT-C 5.1 Toolkit
/*-----
/*  Create : 01.05.98   Update : 11.12.00
/*-----
/*  Descriptions
/*  -----
/*  Inputs      : Reset control [rc]
/*
/*  Outputs     : Instruction acknowledge type [ackfl]
/*
/*  Description :
/*  This function handles the 'API_RESET' instruction to initialize the
/*  FOOBAR hardware and software to an initial state.
/*
*****/

```

# Control Flow 1: Return void if possible

```
AiUInt8 UnivWriteRegister(AiUInt32 RegBaseAddr, AiUInt16 Register, AiUInt32 regValue)
{
    *(AiUInt32*)(Register + RegBaseAddr) = BSWAP32(regValue);
    return TRUE;
}
```

```
AiUInt8 Foo(...)
{
    if (UnivWriteRegister(BoSta[BoNo].BoRegBasAddr, VINT_EN, Mask))
        return TRUE;
    return IntEnErr;
}
```



```
void UnivWriteRegister(AiUInt32 RegBaseAddr, AiUInt16 Register, AiUInt32 regValue)
{
    *(AiUInt32*)(Register + RegBaseAddr) = BSWAP32(regValue);
}
```

```
void Foo(...)
{
    UnivWriteRegister(BoSta[BoNo].BoRegBasAddr, VINT_EN, Mask);
}
```

# Control Flow 2: Use early return

```
foo()  
{  
    if (condition) {  
        <long code>  
    } else {  
        <short code>  
    }  
}
```

```
foo()  
{  
    if (!condition) {  
        <short code>  
        return;  
    }  
    <long code>  
}
```

# Control Flow 3: Use early continue

```
for (...) {  
    if (condition) {  
        <long code>  
    } else {  
        <short code>  
    }  
}
```



```
for (...) {  
    if (!condition) {  
        <short code>  
        continue;  
    }  
    <long code>  
}
```



# Control Flow 4: Avoid if-return-else

```
if (condition) {  
    /* ... */  
    return 1;  
} else {  
    /* ... */  
    return 0;  
}
```

```
if (condition) {  
    /* ... */  
    return 1;  
}  
/* ... */  
return 0;
```

# Debugging Code

Don't clutter the source with trivial debug code once you're done developing.

```
void ReRouter::setName ( String str )
{
    debug.in  ("ReRouter::setName") ;
    config.copyString (name, str) ;
    debug.print (5, "ReRouter::setName: name =", name) ;
    debug.out ("ReRouter::setName") ;
}
```


# C++

- Use STL container templates for lists, maps, vectors.
- Never use 'using namespace'.
- Use bool/true/false.
- If possible, pass arguments by reference.
- Don't put related classes in separate source files. Keep them together (within reason).
- Avoid classeritis!



# Example: Classeritis

```
class MECmdCut : public MECmdClear  
  
class MECmdClear : public MEMultiDirCommand  
  
class MEMultiDirCommand : public MEMultiNodeCommand<MEDirData>  
  
template <class DataType>  
class MEMultiNodeCommand : public MENodeCommand  
  
class MENodeCommand : public GFCommand
```



```
class MECmdCut : public MECmdDelete  
  
class MECmdDelete : public GFCommand
```

# Miscellaneous

- Use const religiously.
- Compile with all warnings on (-Wall) and fix the warnings.
- Premature optimization is the root of all evil.

# How To Stay Sane

# General Guidelines

- Know yourself: discover your strengths and weaknesses.
- Know when you are at your best/worst and plan accordingly.
- The importance of sleep after learning new things.
- The importance of taking a pool/foosball/dart/bathroom/... break.

# Overtime

- Don't, unless:
  - It is of a short duration (< 2 weeks), happens rarely and is to reach a realistic deadline that is tied to a fixed date like a trade show or something similar.
  - You are having fun! But don't neglect your family, friends, social life.
- Structural overtime is pointless: your productivity will remain the same at best, or (more likely) go down. You get tired, cranky, it's harder to think and remain concentrated.
- Seek another job if you are forced to do this.

# Questions?

e-mail:

[hansverk@cisco.com](mailto:hansverk@cisco.com)  
[hverkuil@xs4all.nl](mailto:hverkuil@xs4all.nl)