While working on the Code Archeology talk, presented at ACCU 2010, we realized that the "Physics of Software", and in particular “Code Entropy,” is a topic worth presenting by itself.

By observing how a codebase changes over time you start to notice there are strong and weak forces that shapes the code, and code can be characterized into having a stable or unstable equilibrium. Maintaining and developing a codebase over time is about pushing it in the right direction, always towards higher grounds.

We have studied a large and successful codebase written in C and C++ to find good observations of stable and unstable code, and tried to identify some of the governing forces. We will focus on the small stuff, snippets of code written in C.

A 90 minute session at the ACCU conference
Oxford, April 13-16 2011
(abridged version)
• Introduction
• Results of survey
Codebase Under Discussion

- C and C++
- a few million lines of code
- processor agnostic (TI, Phillips, PPC, ARM, Intel, and more...)
- currently developed and maintained by ~200 developers
- typically 50-200 commits per day
- very visible traces back to '80s and '90s
Examples of past products

1987

1992

1997

2000

2003

2004

2005

2006

2006
Examples of current products
This can only be achieved through:
This can only be achieved through:
• visible and strong architecture
This can only be achieved through:
• visible and strong architecture
• well defined development processes
This can only be achieved through:
- visible and strong architecture
- well defined development processes
- proper documentation of codebase
This can only be achieved through:
• visible and strong architecture
• well defined development processes
• proper documentation of codebase
• corporate coding standards
This can only be achieved through:
• visible and strong architecture
• well defined development processes
• proper documentation of codebase
• corporate coding standards
... so we have been told ...
This can only be achieved through:
• visible and strong architecture
• well defined development processes
• proper documentation of codebase
• corporate coding standards
... so we have been told ...
There are only two kinds of codebases: the ones people complain about and the ones nobody cares about anymore...

(inspired by a similar quote by Bjarne Stroustrup)
Commits between 2010-04-01 and 2011-04-01:
22660 commits (avg 61 per day)
150+ committers (60 devs with more than 100 commits)
Entropy is a measure of how organized or disorganized a system is.

(high entropy) > (low entropy)
Code Entropy
Consider two semantically similar code snippets, A and B.
Code Entropy

Consider two semantically similar code snippets, A and B.

```java
... if (!is_open(socket))
    return false;
else
    return true;
}
```

A
Consider two semantically similar code snippets, A and B.

```c
... if (!is_open(socket))
    return false;
else
    return true;
}

A
```

```c
... if (is_open(socket))
    return true;
else
    return false;
}

B
```
Consider two semantically similar code snippets, A and B. If a group of experts are more likely to change A into B, than vice versa, then code snippet A is less stable.
Consider two semantically similar code snippets, A and B. If a group of experts are more likely to change A into B, than vice versa, then code snippet A is less stable.
Code Entropy

Consider two semantically similar code snippets, A and B. If a group of experts are more likely to change A into B, than vice versa, then code snippet A is less stable.
Consider two semantically similar code snippets, A and B. If a group of experts are more likely to change A into B, than vice versa, then code snippet A is less stable. Hence, A has higher entropy than B.
Consider two semantically similar code snippets, A and B. If a group of experts are more likely to change A into B, than vice versa, then code snippet A is less stable. Hence, A has higher entropy than B.
Dilbert’s demon

[Diagram showing a system labeled A and B with particles moving from A to B.]
Codebases are changing over time
The only valid measurement of code quality: WTFs/minute

Good code.

Bad code.

(reproduced with kind permission of Thom Holwerda)
The health of a codebase is best understood by studying how it changes over time.

Is it deteriorating or improving?
The health of a codebase at a moment in time is less useful.
Survey...

After the introduction, Jon and I showed a collection of code snippet selected from change sets in a real codebase. We asked the audience to vote on which code snippet they believe a group of experts would prefer. I.e., which code snippet is more stable and has lower entropy.
Results of survey
if (!is_open(socket))
    return false;
else
    return true;
}
#1 - negation

```c
... if (!is_open(socket))
    return false;
  else
    return true;
}

if (is_open(socket))
    return true;
else
    return false;
}
```

```c
... if (!is_open(socket))
    return false;
  else
    return true;
}
```
void eventlogputs(const char * string) {
    ...
}

#2 - braces
void eventlogputs(const char * string) {
    ...
}
void eventlogputs(const char * string) {
    ...
}
NetAddr_initAsIPv6(&na.addr, &tests[0].addr, 80, 0);
ret = NetAddr_toString(&na.addr, buf, sizeof(buf), true);
ASSERT(ret && strcmp(buf, "[::1]:80") == 0);
NetAddr_initAsIPv6(&na.addr, &tests[0].addr, 80, 0);
ret = NetAddr_toString(&na.addr, buf, sizeof(buf), true);
ASSERT(ret && strcmp(buf, "[::1]:80") == 0);
#4 - use of temporaries

/* According to CSL API, size must be multiple of 4 (However, looking at CSL implementation, it handles non-multiple of 4. Best to follow doc...) */
size = ((size + 3) / 4) * 4;

tmp = size & 0x3;
if (tmp)
    size += (4 - tmp);
#4 - use of temporaries

/* According to CSL API, size must be multiple of 4 (However, looking at CSL implementation, it handles non-multiple of 4. Best to follow doc...) */
size = ((size + 3) / 4) * 4;

/* According to CSL API, size must be multiple of 4 (However, looking at CSL implementation, it handles non-multiple of 4. Best to follow doc...) */
tmp = size & 0x3;
if (tmp)
    size += (4 - tmp);
if (getValuespaceType(valuespace_elem) == PVAL_CUIL_E164) {
    snprintf(verify, sizeof(verify), "onchange=verify_e164('%s',this)", szPath);
} else {
    strcpy(verify, " ");
}

if (getValuespaceType(valuespace_elem) == PVAL_CUIL_E164) {
    snprintf(verify, sizeof(verify), "onchange=verify_e164('%s',this)", szPath);
} else {
    strcpy(verify, " ");
}
if (getValuespaceType(valuespace_elem) == PVAL_CUIL_E164) {
    snprintf(verify, sizeof(verify), "onchange=verify_e164('%s',this)", szPath);
} else {
    strcpy(verify, " ");
}
SIP_bytes2hexstr(bytes, sizeof(bytes), cnonce, len);

SIP_bytes2hexstr(bytes, sizeof(bytes), cnonce, len);
SIP_bytes2hexstr(bytes, sizeof(bytes), cnonce, len);

#6 - sizeof

SIP_bytes2hexstr(bytes, sizeof(bytes), cnonce, len);
#7 - casting

```c
assert((size_t)len < sizeof(cmdname));
```

```c
assert(len < (int)sizeof(cmdname));
```
assert(len < (int)sizeof(cmdname));

assert(((size_t)len < sizeof(cmdname));
#8 - typedef structs or not

```
pPriv->FBufsize = sizeof(struct AUD_STREAM_DATA_STRUCT);
```

```c
pPriv->FBufsize = sizeof(struct AUD_STREAM_DATA_STRUCT);
```
#8 - typedef structs or not

```c
pPriv->FBufsize = sizeof(struct AUD_STREAM_DATA_STRUCT);
```

```c
pPriv->FBufsize = sizeof(AUD_STREAM_DATA_STRUCT);
```
if (priv->vcfsm.Spec != SYSTEM && priv->vcfsm.Spec != NO_SPECIE) {
    fsm_sendMsg(proc, VIDEOEXEC_PROCESS_GRAPH_REJ, priv->vcfsm, NULL);
}

if ((priv->vcfsm.Spec != SYSTEM) && (priv->vcfsm.Spec != NO_SPECIE)) {
    fsm_sendMsg(proc, VIDEOEXEC_PROCESS_GRAPH_REJ, priv->vcfsm, NULL);
}
if (priv->vcfsm.Spec != SYSTEM && priv->vcfsm.Spec != NO_SPECIE) {
    fsm_sendMsg(proc, VIDEOEXEC_PROCESS_GRAPH_REJ, priv->vcfsm, NULL);
}

if (((priv->vcfsm.Spec != SYSTEM) && (priv->vcfsm.Spec != NO_SPECIE)) {
    fsm_sendMsg(proc, VIDEOEXEC_PROCESS_GRAPH_REJ, priv->vcfsm, NULL);
}
static void html_select_option(
    char * dest,
    CUIL_ELEMENT * elem,
    char const * selected,
    char const * referencePath,
    char const * elemname,
    size_t destsize)
{
}

static void html_select_option(
    char * buffer,
    CUIL_ELEMENT * elem,
    char const * selected,
    char const * referencePath,
    char const * elemname,
    size_t buffer_size)
{
}
static void html_select_option(
    char * dest,
    CUIL_ELEMENT * elem,
    char const * selected,
    char const * referencePath,
    char const * elemname,
    size_t destsize)
{

static void html_select_option(
    char * buffer,
    CUIL_ELEMENT * elem,
    char const * selected,
    char const * referencePath,
    char const * elemname,
    size_t buffer_size)
{
#11 - strcpy vs sprintf

```c
strcpy(tmp, gctx->digest.algorithm);
```

```c
sprintf(tmp, "%s", gctx->digest.algorithm);
```

↑  ↓
#11 - strcpy vs sprintf

```c
strcpy(tmp, gctx->digest.algorithm);
```

```c
sprintf(tmp, "%s", gctx->digest.algorithm);
```
static bool isin(const char * needle,
	char * const * haystack)
{
    while (*haystack) {
        if (strcmp(needle, *haystack) == 0) {
            return true;
        }
        ++haystack;
    }
    return false;
}
static bool isin(const char * needle,
                char * const * haystack)
{
    while (*haystack) {
        if (strcmp(needle, *haystack) == 0) {
            return true;
        }
        ++haystack;
    }
    return false;
}

static bool isin(const char * needle,
                 char * const * haystack)
{
    while (*haystack) {
        if (strcmp(needle, *haystack) == 0) {
            return true;
        }
        ++haystack;
    }
    return false;
}
static void cleanup(char ** strings)
{
    char ** r = strings;
    while (*r) {
        free(*r);
        ++r;
    }
    free(strings);
}
static void cleanup(char ** strings)
{
    char ** r = strings;
    while (*r) {
        free(*r);
        ++r;
    }
    free(strings);
}
void popRemoveRoute(pop_buffer_t *popBuffer, pop_t *pop)
{
  parg_t *pa;
  FSMADDR fsm;
  int gate_id = GATE_UNDEFINED;
void popRemoveRoute(pop_buffer_t *popBuffer, pop_t *pop)
{
    parg_t *pa;
    FSMADDR fsm;
    int gate_id = GATE_UNDEFINED;
#15 - fall through

```c
case H263:
    if (!ttvenc->add_payload_headers)
        skip = 4 * ((data[0] >> 6) == 2) + 4 * ((data[0] >> 6) == 3);
        // intentional fall through
    case H261:
        if (!ttvenc->add_payload_headers) {
            skip += 4;
```
case H263:
    if (!ttvenc->add_payload_headers)
        skip = 4 * ((data[0] >> 6) == 2) + 4 * ((data[0] >> 6) == 3);
    // intentional fall through

case H261:
    if (!ttvenc->add_payload_headers) {
        skip += 4;
    }
#16 - sizeof

globData->Time = malloc(sizeof *globData->Time);

globData->Time = malloc(sizeof(TIME_STR));
globData->Time = malloc(sizeof *globData->Time);

globData->Time = malloc(sizeof(TIME_STR));
typedef struct {
    FSMADDR portHandler; /*receiver of serial visca commands*/
    bool hasVisca;
    bool narrowedTiltRange;
    bool brightnessGradient;
} CAM_Init_Req_Struct;

struct CAM_Init_Req_Struct {
    FSMADDR portHandler; /*receiver of serial visca commands*/
    bool hasVisca;
    bool narrowedTiltRange;
    bool brightnessGradient;
};
typedef struct {
    FSMADDR portHandler; /*receiver of serial visca commands*/
    bool hasVisca;
    bool narrowedTiltRange;
    bool brightnessGradient;
} CAM_Init_Req_Struct;

struct CAM_Init_Req_Struct {
    FSMADDR portHandler; /*receiver of serial visca commands*/
    bool hasVisca;
    bool narrowedTiltRange;
    bool brightnessGradient;
};
if ( 0 != strcmp(argv[3], "debug")) {
    con_puts(co, "Unknown ctx command\n");
    return PARAMETER_ERROR;
}
if ( 0 != strcmp(argv[3], "debug")) {
    con_puts(co, "Unknown ctx command\n");
    return PARAMETER_ERROR;
}
#19 - size_t and naming

```c
char* SIP_bytes2hexstr(const unsigned char* input, int len, char* output, int maxlen);
```

```c
char* SIP_bytes2hexstr(const unsigned char* src, size_t srclen, char* dest, size_t destsize);
```
#19 - size_t and naming

char* SIP_bytes2hexstr(const unsigned char* src, size_t srclen, char* dest, size_t destsize);
#include "ccf.h"
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
Appendix
Significant events (recently)

- Warning free (since 2007)
- genbuild2 (2005)
- Automatic Continuous Integration (2005)
- CVS to SVN (2006)
- C++ introduced (~2006)
- RCS to CVS (2002)
- Unit Tests (2005)
- Proper C and C++ training (since 2007)
- Module Tests (2007)
- Stopped using code generating tools (2007)
- Major directory reorganization (2008)
- CVS to SVN (2006)
- Automatic system testing (since ~2007)
Maintrunk and branching strategy
Key observations from the Lysaker codebase 2010-2011

- readability
- compilation time
- dependency analysis / control / management
- less use of inline functions
- dependency injection
- more use of const, size_t and assert
- more use of forward declarations
- typedef struct deprecated
- moving away from corporate libraries
- using new language features, eg C99
- focus on standard C
- prefer C over C++, only use C++ where necessary
- proper casting, eg, \(\text{len} < \text{(int)sizeof x} \) VS (size_t)\(\text{len} < \text{sizeof x}\)
- macros gradually replaced, less use of preprocessor
- peer reviews / pair programming / patches mailing list
- collective ownership
- towards idioms
and some more observations

• change functions to return void callers do not care about the return value
• xx_assert() replaced with assert()
• reducing scope of variables
• abbreviations replaced with more descriptive names
• explanation variables
• initialization of variables
• for (int i=0; i<42; i++) vs for (int i=0; i!=42; ++i)
• reduce use of fix ints (uint32_t -> int)
• order of include files
• early returns, less nesting
• intention revealing typenames (eg, WORD -> bool)
• less use of NULL
• log messages tend to be removed
and, even more observations

• dehungarization
• decamelization
• aligned braces for functions, disaligned for if/for/while
• {} removed from single line blocks
• removing parenthesis
• long lines are broken into 80 character lines
• tabs are replaced with spaces
• char * s seems to be more stable than K&R and BS style
• In C, post-increment seems to be more stable than pre-increment (i++ vs ++i)
• focus on “robust” layout
• increased horizontal and vertical spacing
• indentation 4 spaces
• space around operators
• block comments are removed
• removing comments by improving code