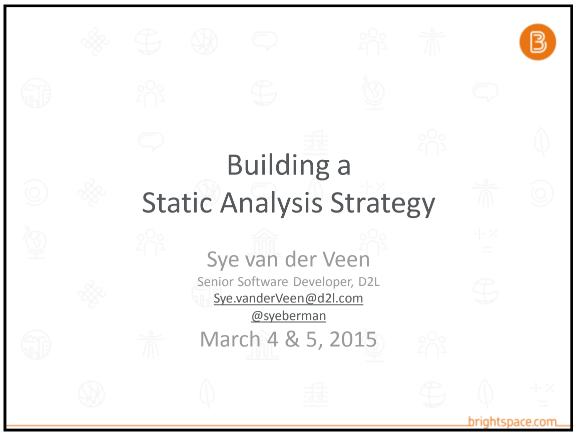
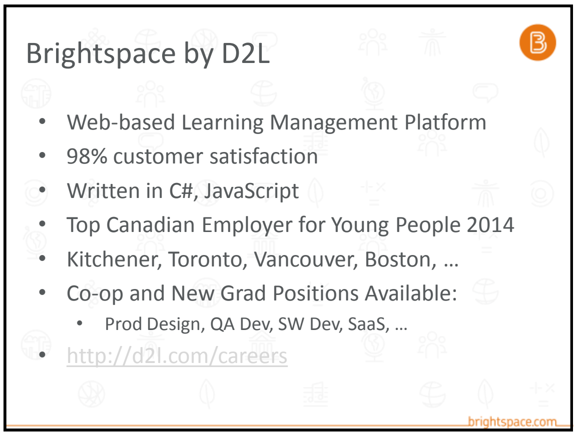


Building a Static Analysis Strategy

Sye van der Veen
Senior Software Developer, D2L
Sye.vanderVeen@d2l.com
@syeberman

March 4 & 5, 2015

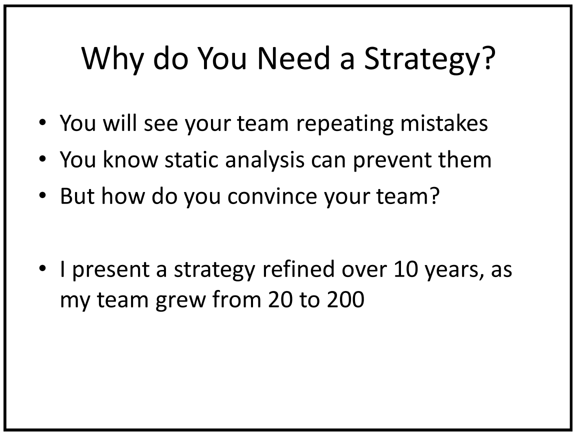
brightspace.com



Brightspace by D2L

- Web-based Learning Management Platform
- 98% customer satisfaction
- Written in C#, JavaScript
- Top Canadian Employer for Young People 2014
- Kitchener, Toronto, Vancouver, Boston, …
- Co-op and New Grad Positions Available:
  - Prod Design, QA Dev, SW Dev, SaaS, …
- http://d2l.com/careers

brightspace.com



Why do You Need a Strategy?

- You will see your team repeating mistakes
- You know static analysis can prevent them
- But how do you convince your team?

- I present a strategy refined over 10 years, as my team grew from 20 to 200

Static Analysis Strategy

- Ignore all compiler warnings

`warning: function "Append" declared implicitly`

main.c

```
void main(void)
{
  List *vals = ListNew(20);
  int i;
  for(i=0; i<5; i++) {
    Append(vals, i);
  }
}
```

list.c

```
void Append(List *x, Obj *y)
{
  x[x->len] = y;
  x->len++;
}
```

Static Analysis Strategy

- ~~Ignore all compiler warnings~~
- Treat all compiler warnings as errors

error: unreferenced formal parameter "future"

```
void Initialize(void *future)
{
  State = malloc(sizeof(State_t));
  memset(State, 0, sizeof(State_t));
  // future not currently used
}
```

Static Analysis Strategy

- ~~Ignore all compiler warnings~~
- ~~Treat all compiler warnings as errors~~
- Treat recommended warnings as errors

warning: "y" set but never used

Useful in a small projects

```
int x, y;
int max = 0;

for(x=0; x<Len(array); x++)
{
  y = Item(array, x);
  if(x>max) max = x;
}
printf("Max is %d\n", max);
```

Not in large ones

```
int x, y;
int result = SUCCESS;

for(x=0; x<MAX_ANTS; x++)
{
  y = StartAnt(x); // cannot fail
#ifdef BUG_TABLE_AVAILABLE
  // Register ants w/bug table
  result = RegisterBug(y);
  if(result!=SUCCESS) break;
#endif
}
return result;
```

Static Analysis Strategy

- ~~Ignore all compiler warnings~~
- ~~Treat all compiler warnings as errors~~
- ~~Treat recommended warnings as errors~~
- Treat all compiler warnings as errors…to start
  - Then disable individual warnings based on project

Static Analysis Strategy

- Build errors
  - Treat all compiler warnings as errors
- Suppressed warnings
  - Disable individual warnings based on project

PC-Lint

- Fantastically fast
- …but not the *best* tool for finding bugs based on the values of variables/arguments/etc
  - It does an OK job, sometimes

warning 665: Unparenthesized parameter in macro passed expression

```
#define MULT(a,b) (a*b)
int AdjustGain(int raw)
{
  return MULT(100, 4+raw);
}
// Bug: AdjustGain(20) --> 420?!
```

info 833: 'increment' is typed differently in another module

ants.c

```
static void
  increment(Ant *x, int y)
{
  x->kills += y;
}
```

bugs.c

```
static void
  increment(Bug *x)
{
  x->pupae += 1;
}
```

Static Analysis Strategy

- Build errors
  - Treat all compiler warnings as errors
  - Enable PC-Lint "-w2" in build
- Suppressed warnings
  - Disable individual warnings based on project

info 768: global struct member 'kids' not referenced

ants.h

```
#define MAX_KIDS 600
typedef struct _Ant {
  int kills;
  char *name;
  struct _Ant *kids[MAX_KIDS];
} Ant;
```

ants.c

```
int AntKills(Ant *x) {
  return x->kills;
}
void AntSetKills(Ant *x, int k) {
  x->kills = k;
}
const char *AntName(Ant *x) {
  return x->name;
}
void AntSetName(Ant *x, char *n) {
  x->name = n;
}
```

If You Remember One Slide Today…

- Static analysis is great for two reasons
  - It points out where bugs may be lurking today
    - Catch these as early as possible by making them errors
  - It suggests areas that may be difficult to maintain
    - Set these aside for when you can spare the time

"Set Aside?"

- Either enable these extra warnings using a special flag to your build process
- Or have a separate server that runs with extra warnings and displays them on a website
- Or create a script that emails your team one random warning a day
- Or award points when warnings are fixed
- Or…

Coverity and Klocwork

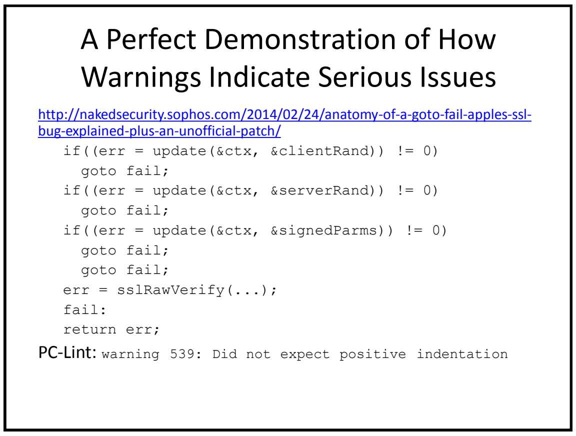
- Fantastic at finding value-based and inter-function bugs
- Very much slower than PC-Lint
- Quite a lot more expensive than PC-Lint
  - You may not be able to afford giving everyone in the team a license
- As such, these warnings normally set aside

Static Analysis Strategy

- Build errors
  - Treat all compiler warnings as errors
  - Enable PC-Lint "-w2" in build (and others you find important)
- Temporarily-ignored warnings
  - Set aside for when you can spare the time
- Suppressed warnings
  - Disable individual warnings based on project

Sources of Fatigue

- You'll start with a large backlog of issues
  - Mark the entire backlog as "ignored"
  - Or hide the master list from developers
- The list will never be empty
  - Set realistic goals ("# warnings / # files < 2")
  - Or encourage developers to mark as "ignored"
- False positives will feel like wasted time
  - Disable warnings prone to false positives
  - Or use as clues to code that should be rewritten for clarity
- Developers may say they don't have time
  - Create a team dedicated to fixing warnings
  - Or demonstrate how warnings indicate serious issues

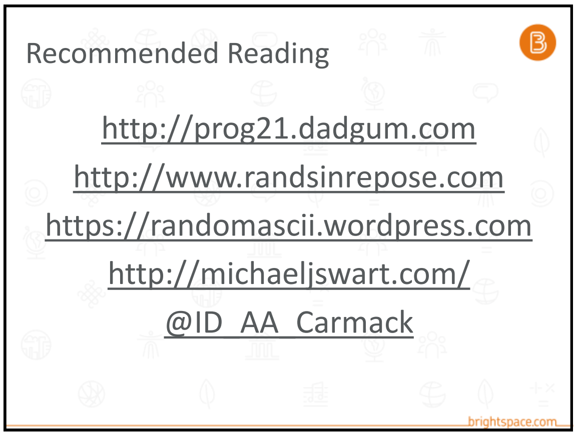


A Perfect Demonstration of How Warnings Indicate Serious Issues

http://nakedsecurity.sophos.com/2014/02/24/anatomy-of-a-goto-fail-apples-ssl-bug-explained-plus-an-unofficial-patch/

```
if((err = update(&ctx, &clientRand)) != 0)
  goto fail;
if((err = update(&ctx, &serverRand)) != 0)
  goto fail;
if((err = update(&ctx, &signedParms)) != 0)
  goto fail;
  goto fail;
err = sslRawVerify(...);
fail:
return err;
```

PC-Lint: `warning 539: Did not expect positive indentation`



Recommended Reading

http://prog21.dadgum.com

http://www.randsinrepose.com

https://randomascii.wordpress.com

http://michaeljswart.com/

@ID_AA_Carmack

brightspace.com