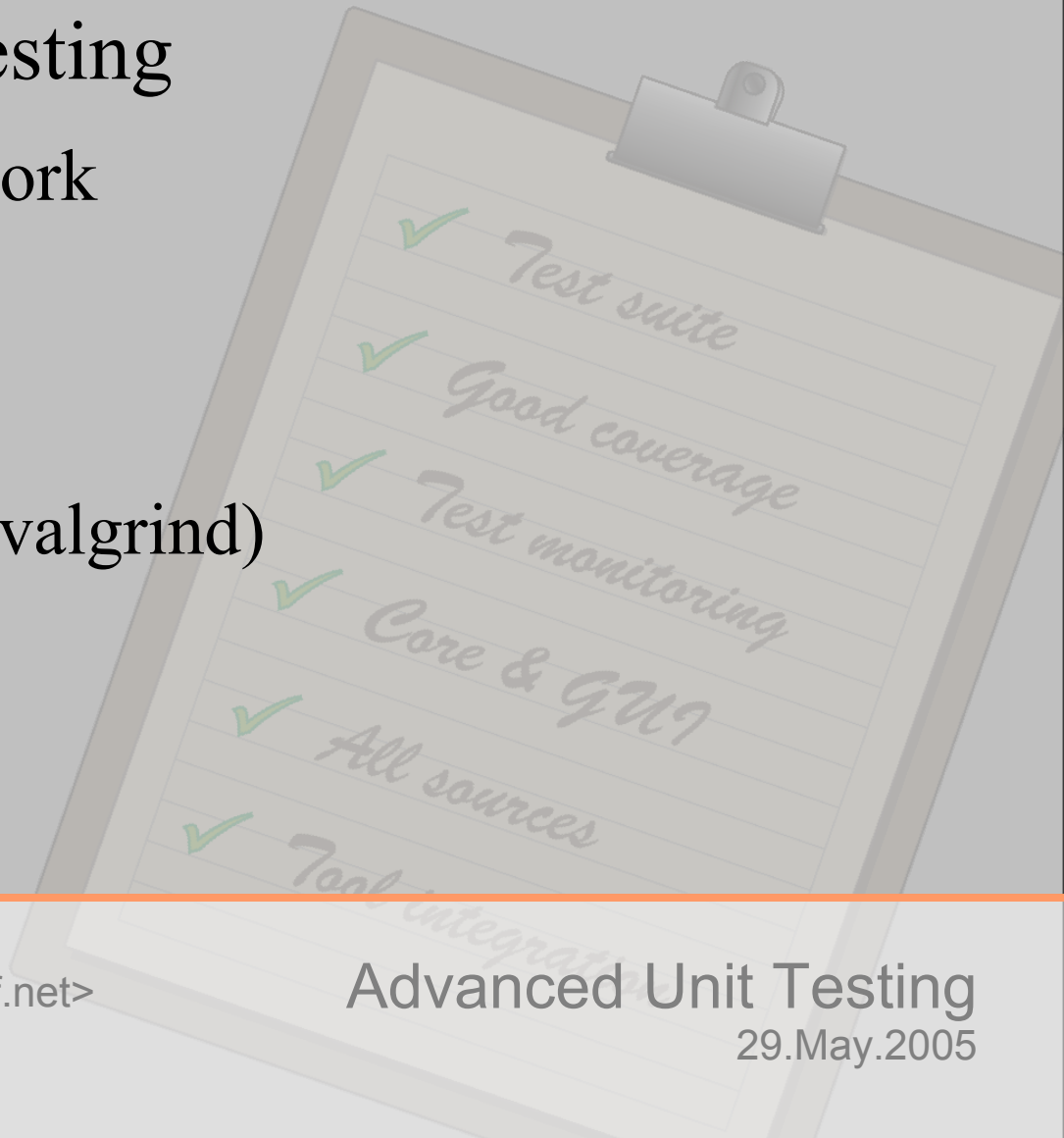
Advanced Unit Testing

Thomas Wabner <waffel@users.sf.net>
Stefan Kost <ensonic@users.sf.net>

Outline

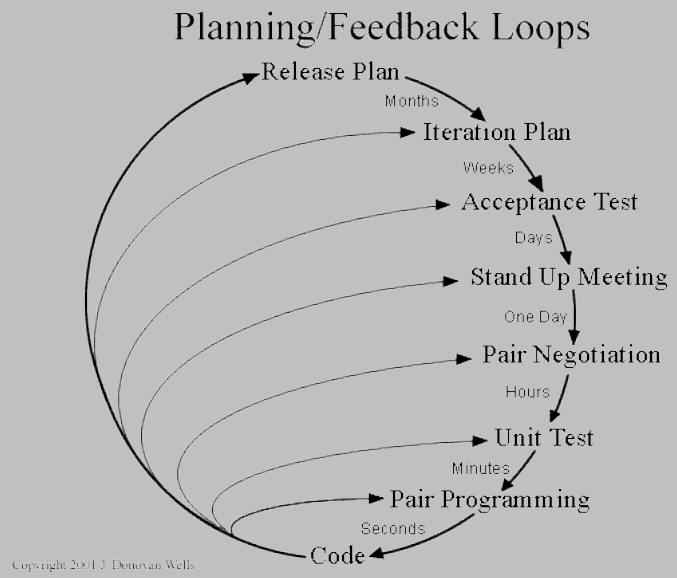
- Motivation
- Introduction to Unit Testing
  - Check Testing Framework
  - Writing tests
- Advanced Testing
  - Tool integration (lcov, valgrind)
- Quality assurance
- Conclusion



Thomas Wabner <waffel@users.sf.net>
Stefan Kost <ensonic@users.sf.net>

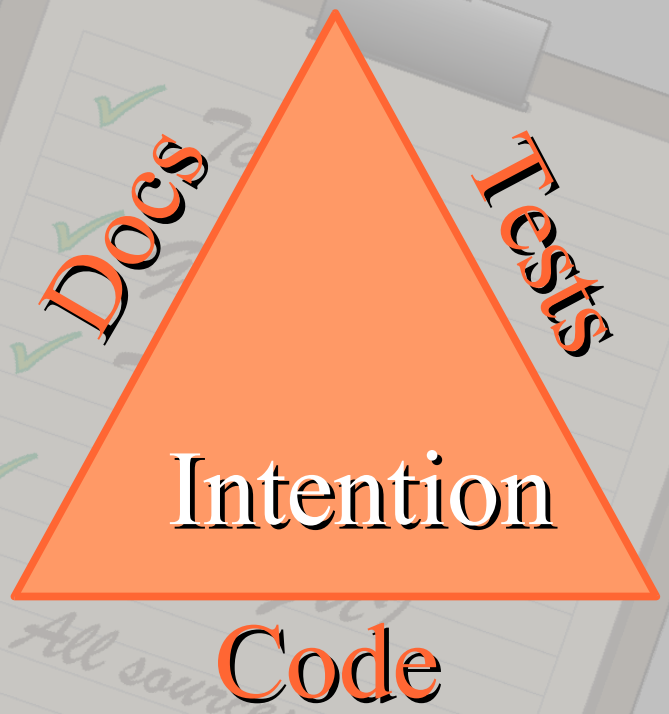
Motivation

- Software Quality
- Many free tools available
- How to integrate into Development Process



Introduction

- Extreme Programming
- Unit Testing = Automated Testing
  - Verify intention of code
  - Current code still works after a change
  - New code works the way it is supposed to

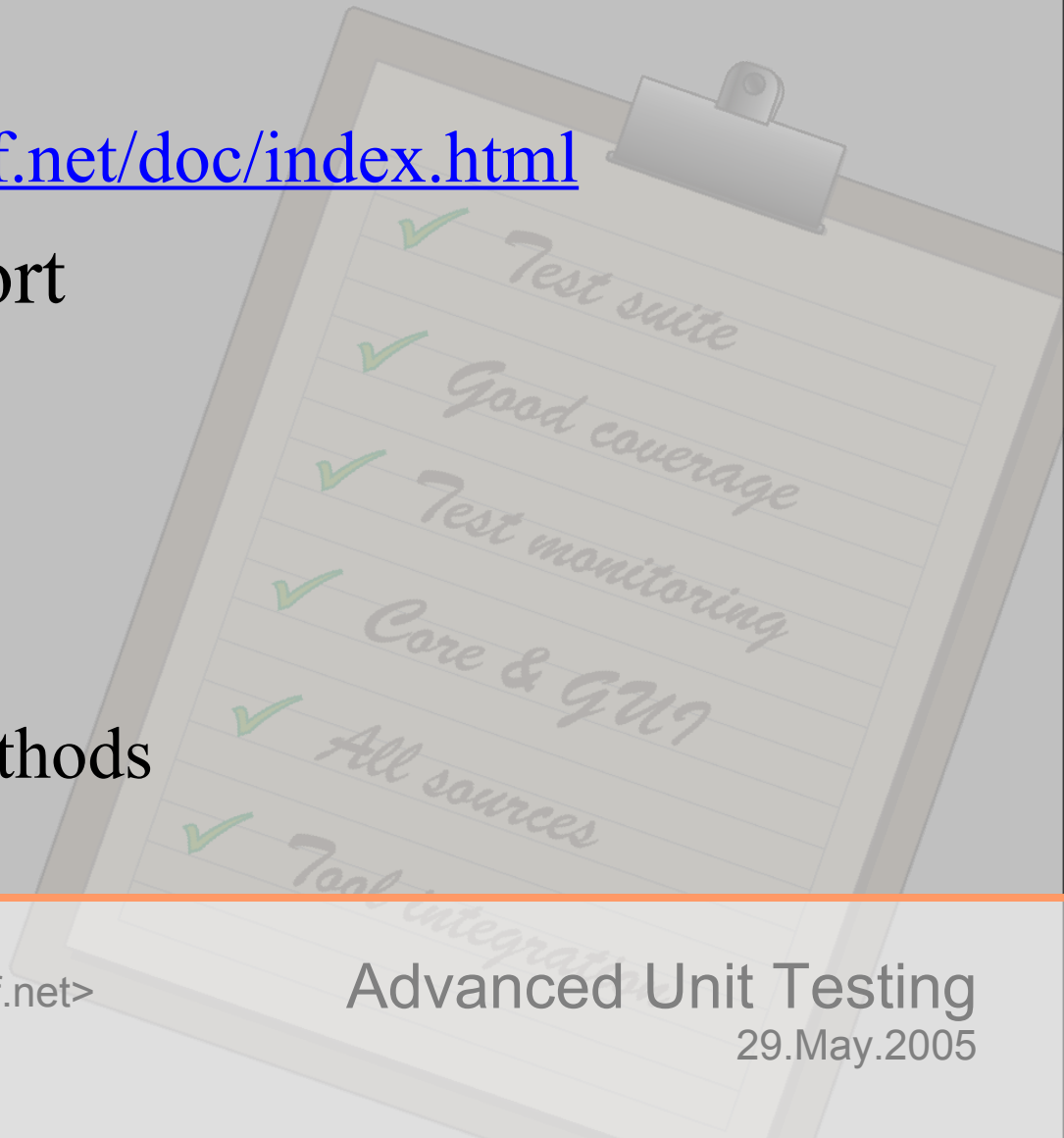


Autotools

- Automake provides test suite integration
- => high-level test support
- 'make check' runs tests
- Tests are programs or scripts
  - Return code = 0: okay
  - Return code > 0: failure

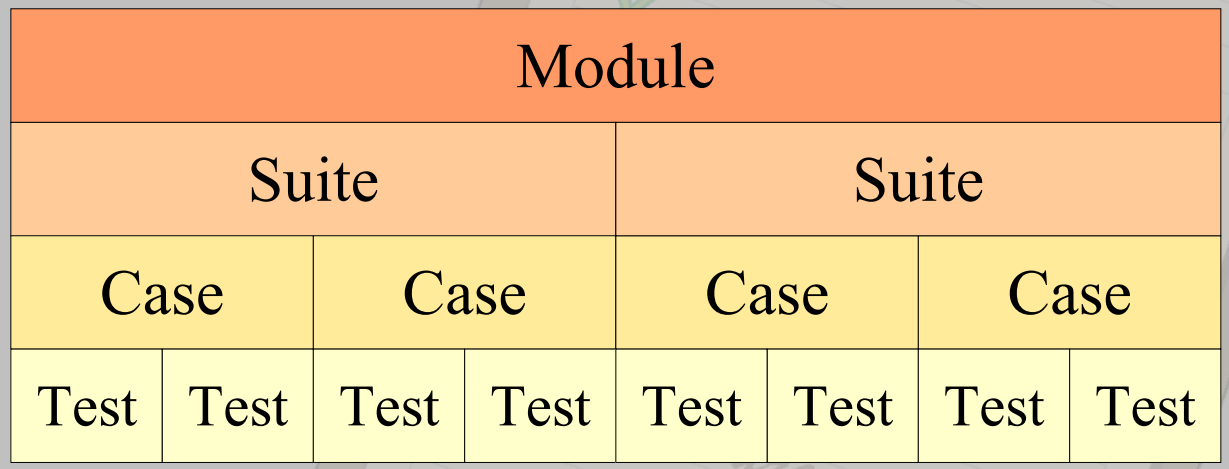
Check

- Framework for C (similar to jUnit for Java)
  - http://check.sf.net
  - Tutorial: http://check.sf.net/doc/index.html
- => low level test support
- Features
  - Sandbox
  - Logging
  - Many useful helper methods



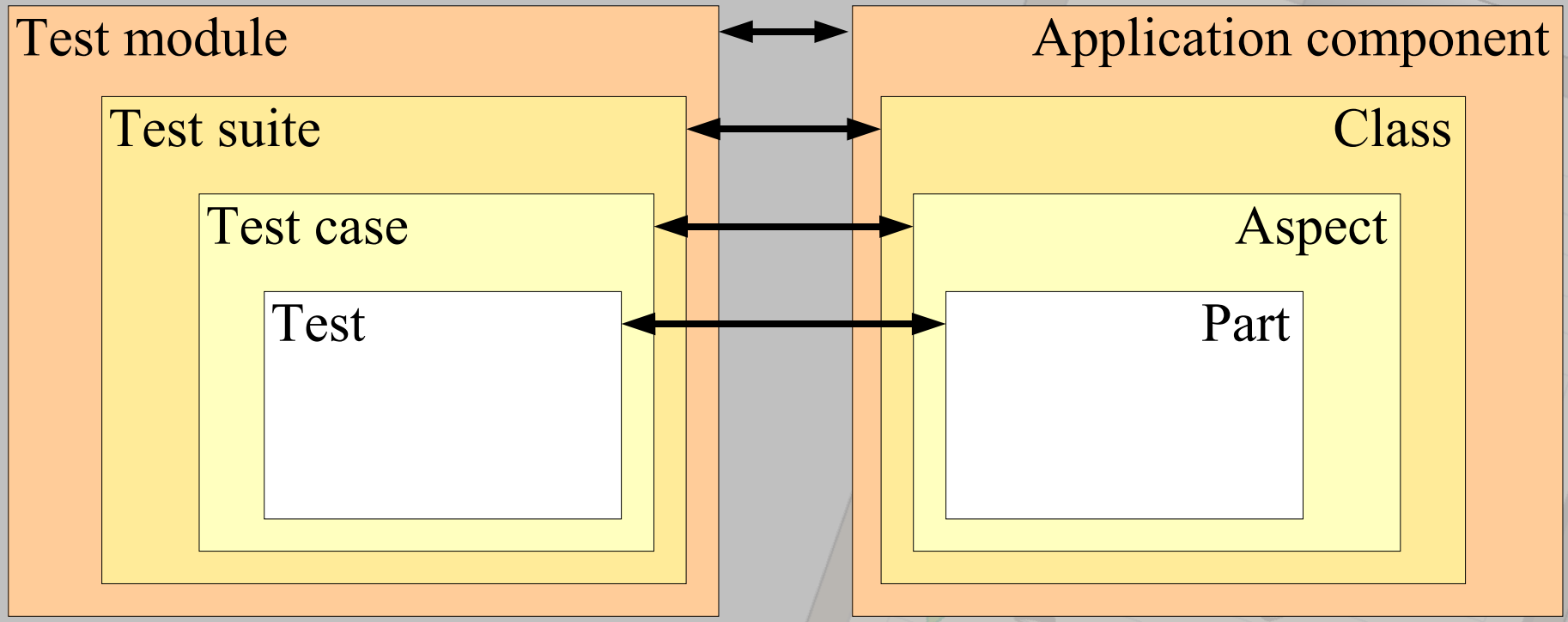
Check

- Tests are structured:
  - Test module (binary)
  - Test suite(s)
  - Test case(s)
  - Test(s)



Check

- Applications are structured too
- Map test and application structure



Test Module

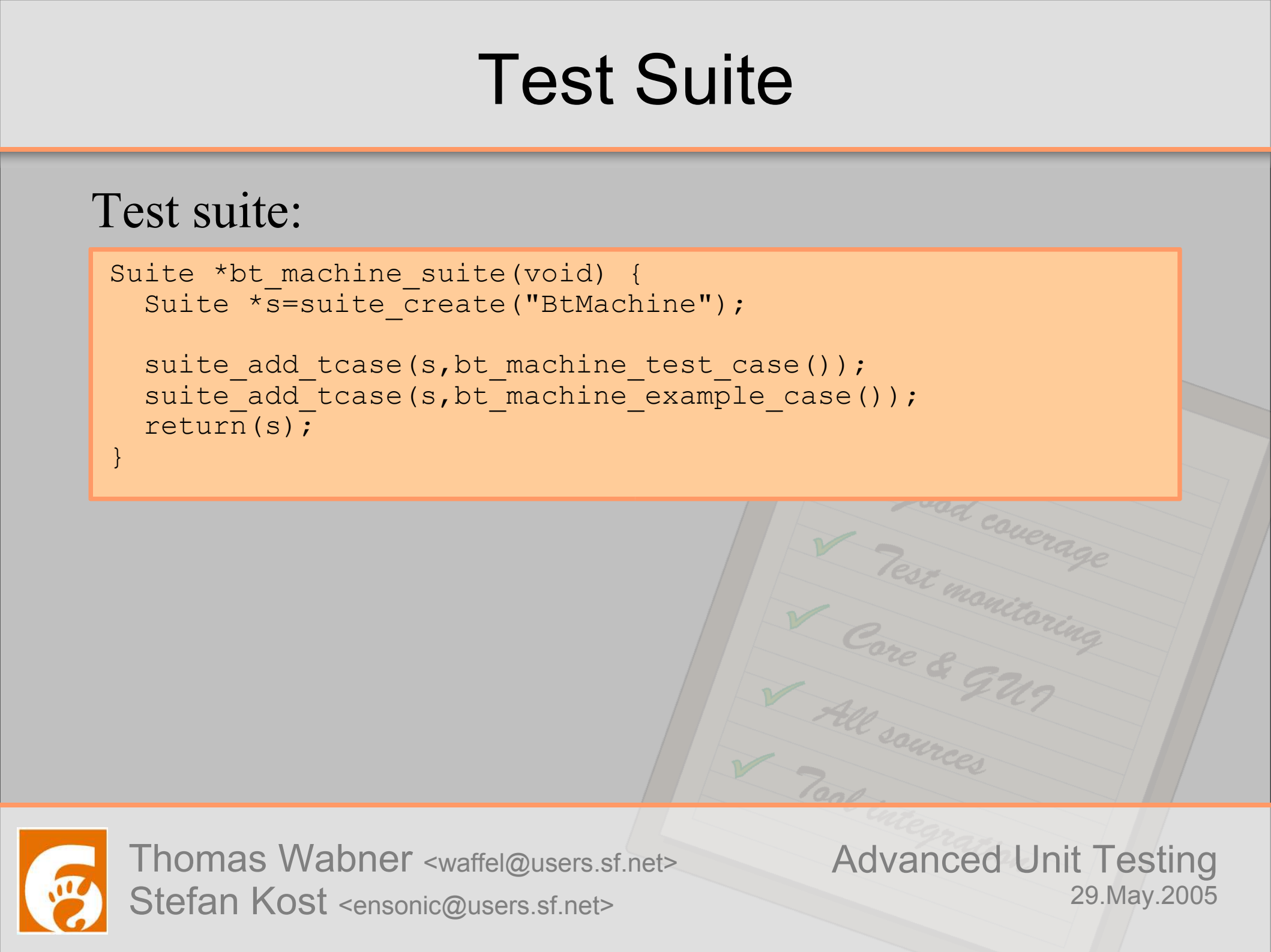
Test module:

```
/* start the test run */
int main(int argc, char **argv) {
  int number_failed;
  SRunner *sr;

  sr=srunner_create(bt_core_suite());
  srunner_add_suite(sr, bt_machine_suite());
  ...

  srunner_run_all(sr,CK_NORMAL);
  number_failed=srunner_ntests_failed(sr);
  srunner_free(sr);

  return(number_failed==0) ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

Test Suite

Test suite:

```
Suite *bt_machine_suite(void) {
  Suite *s=suite_create("BtMachine");

  suite_add_tcase(s,bt_machine_test_case());
  suite_add_tcase(s,bt_machine_example_case());
  return(s);
}
```



Test Case

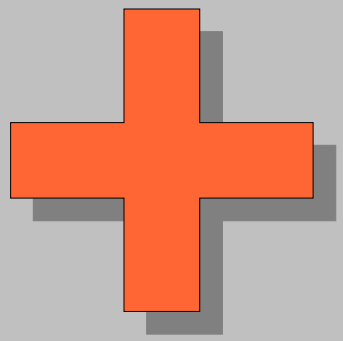
Test case:

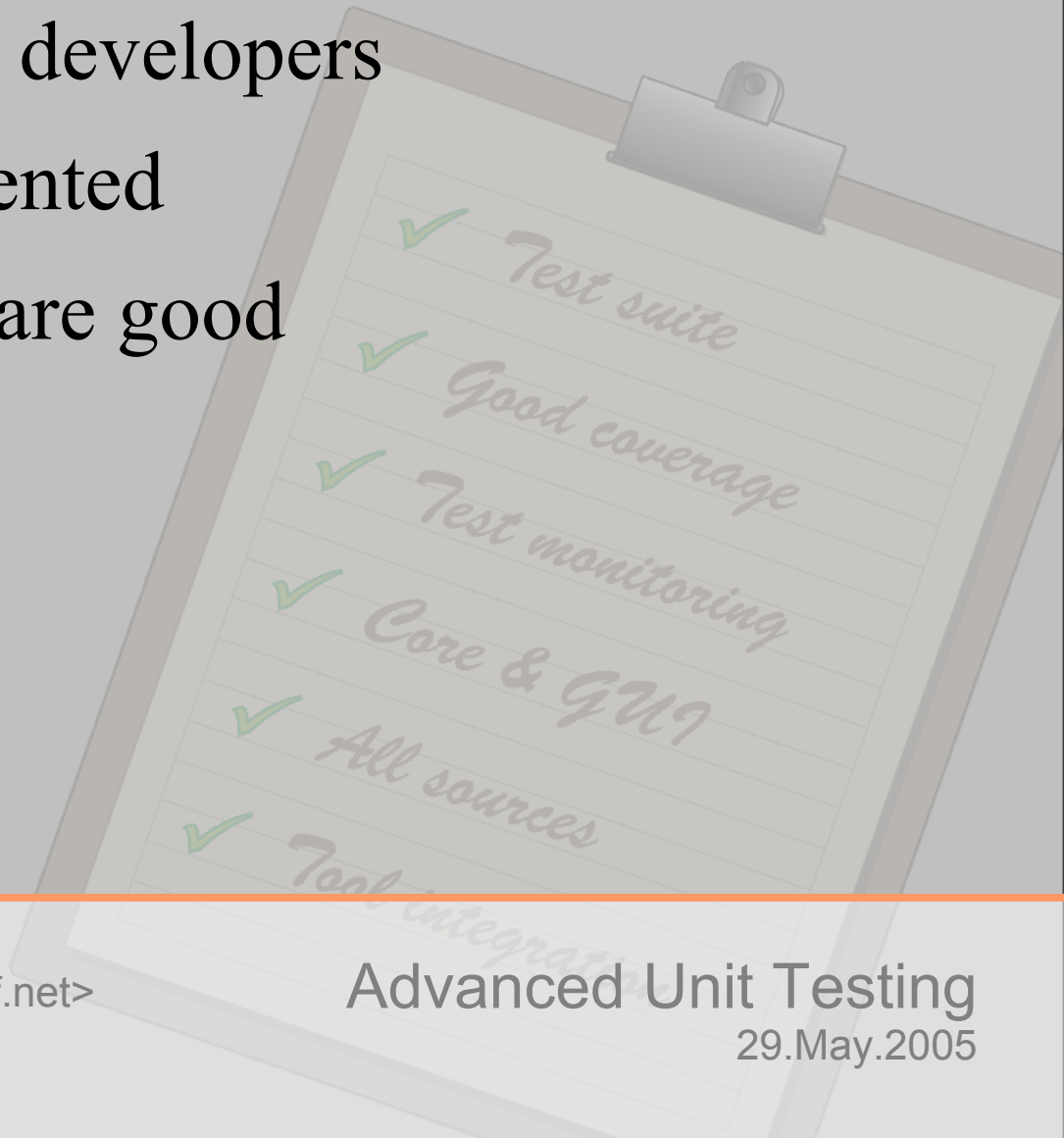
```
TCase *bt_machine_example_case(void) {
  TCase *tc = tcase_create("BtMachineExamples");

  tcase_add_test(tc,test_btmachine_insert_input_level1);
  tcase_add_test(tc,test_btmachine_insert_input_level2);
  tcase_add_unchecked_fixture(tc, test_setup, test_teardown);
  return(tc);
}
```

Thomas Wabner <waffel@users.sf.net>
Stefan Kost <ensonic@users.sf.net>

Positive Tests

- API usage examples
- Serves as a HowTo for developers
- Should be well documented
- Ensures that API docs are good





Positive Tests

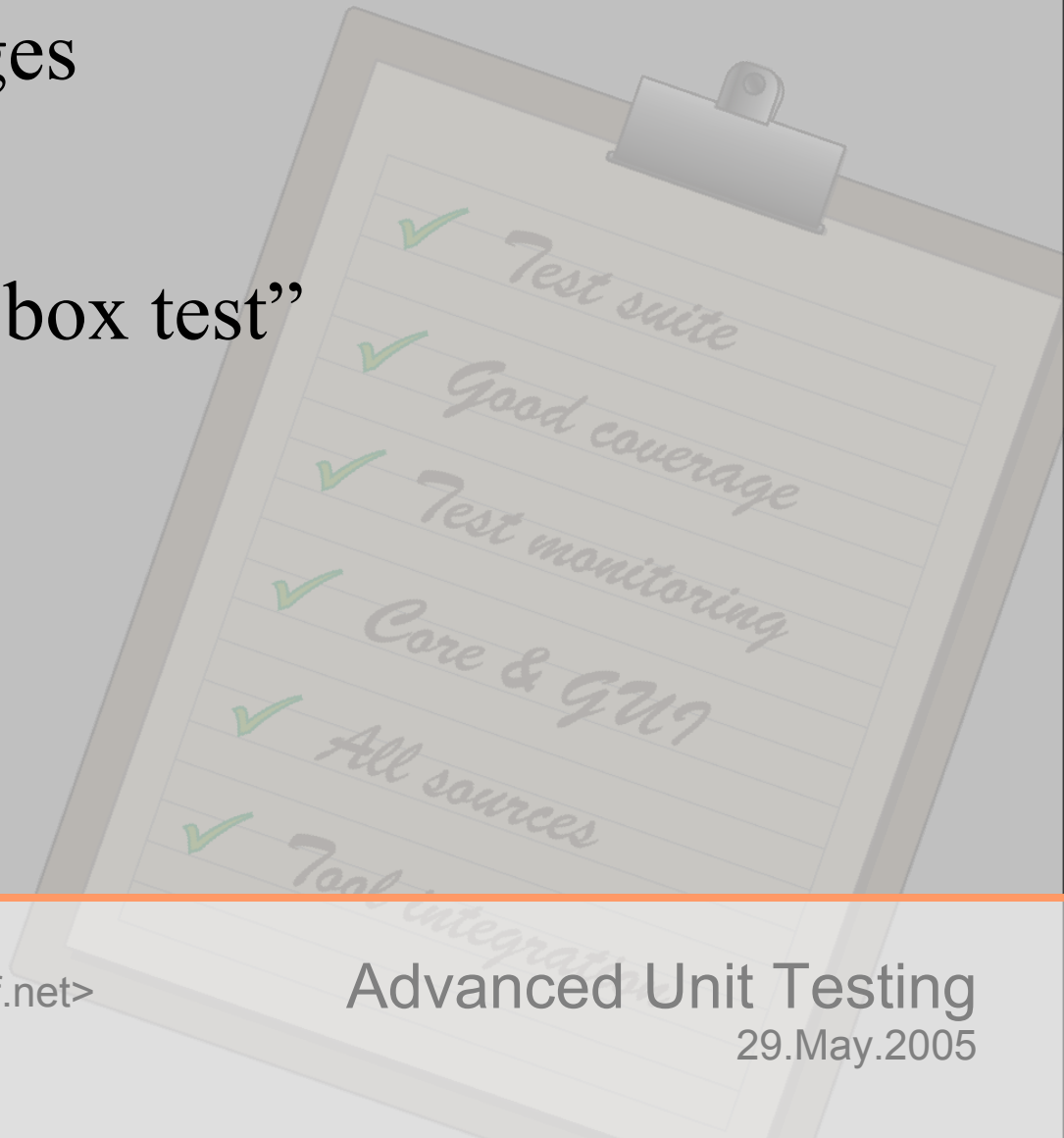
Test:

```
START_TEST(test_btsourcemachine_obj1){
  BtSourceMachine *machine=NULL;

  ...

  /* try to create a source machine */
  machine=bt_source_machine_new(song,"gen","sinesrc",1);
  fail_unless(machine!=NULL, NULL);

  ...
}
END_TEST
```

Thomas Wabner <waffel@users.sf.net>
Stefan Kost <ensonic@users.sf.net>

Negative Tests

- Try to break the API
- Testing parameter ranges
- Testing program flow
- Also known as “Black box test”



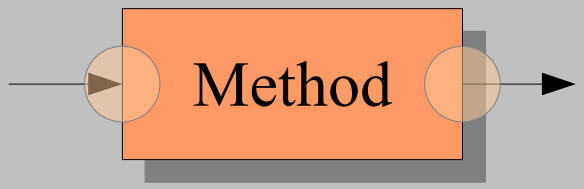
Negative Tests

Test:

```
START_TEST(test_btsourcemachine_obj1){
  BtSourceMachine *machine=NULL;

  ...

  /* try to create a source machine with wrong plugin name */
  machine=bt_source_machine_new(song,"id","nonsense",1);
  fail_unless(machine==NULL, NULL);

  ...
}
END_TEST
```

Pre- and Post-Conditions

- Pre-Conditions
  - API should check that it is used properly
- Post Conditions
  - API should check that it returns sane values
- We need to test that this is implemented



Thomas Wabner <waffel@users.sf.net>
Stefan Kost <ensonic@users.sf.net>

Pre-Conditions

Test:

```
check_init_error_trap("bt_setup_new","BT_IS_SONG(song)");
setup=bt_setup_new(NULL);
fail_unless(check_has_error_trapped(), NULL);
```

Code:

```
BtSetup *bt_setup_new(const BtSong *song) {
  g_return_val_if_fail(BT_IS_SONG(song),NULL);
```

Thomas Wabner <waffel@users.sf.net>
Stefan Kost <ensonic@users.sf.net>

Post-Conditions

Test:

```
check_init_error_trap(
   "bt_machine_get_global_param_index","(error && *error)");
res=bt_machine_get_global_param_index(machine,"x$%&",&error);
fail_unless(check_has_error_trapped(), NULL);
```

Code:

```
glong bt_machine_get_global_param_index(
   const BtMachine *self, const gchar *name, GError **error) {
   ...
   g_assert((found || (error && *error)));
```

GObject Reference Counts

- GObject uses reference counting
- Errors:
  - Unref too often
    -> will probably segfault
  - Forget to unref an instance when done
    -> causes a memory leak
- Useful tool: RefDbg
  - http://refdbg.sourceforge.net/

GObject Reference Counts

Test:

```
BtApplication *app;
...
g_object_checked_unref(app);
```

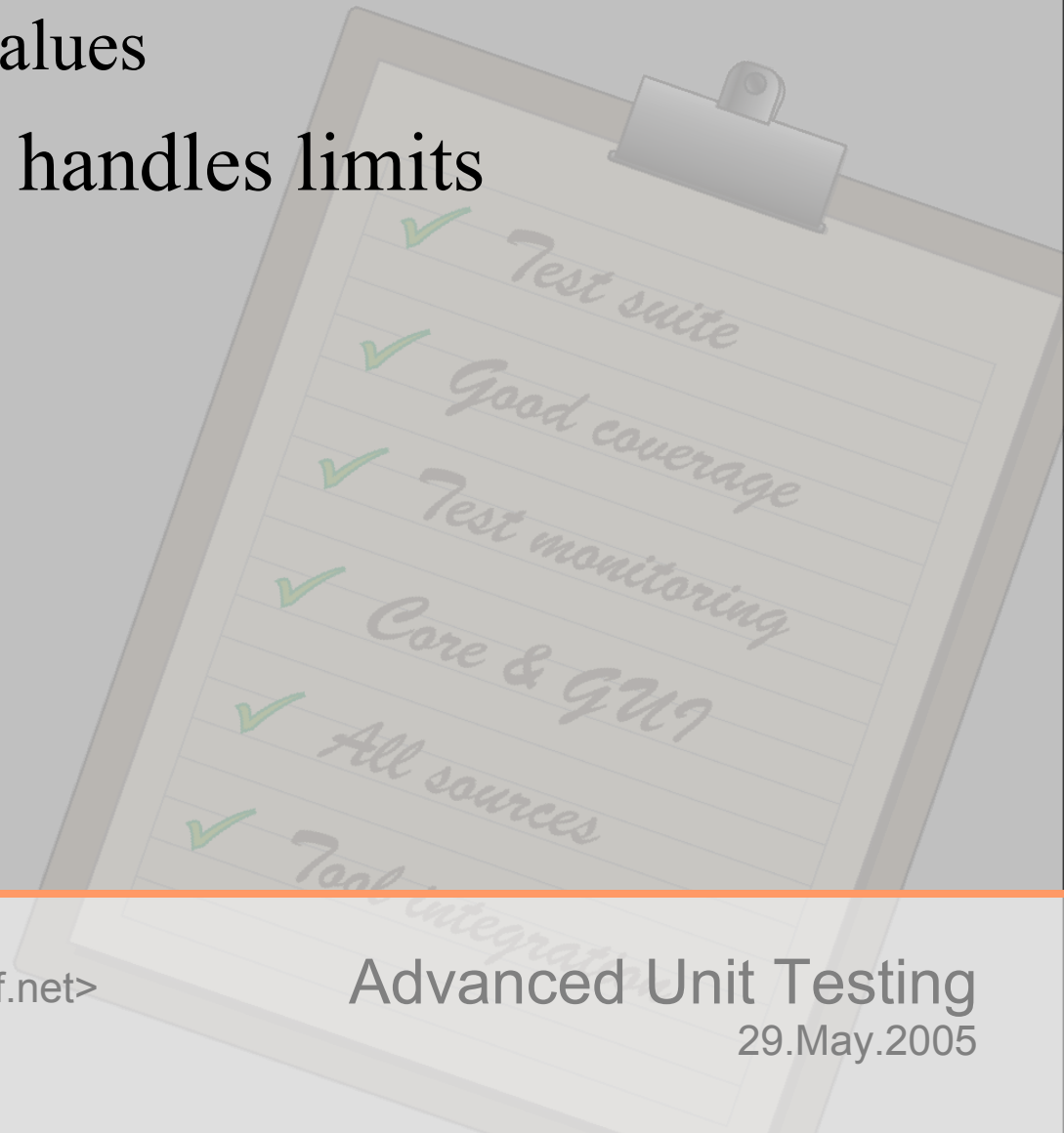
Implementation:

```
void g_object_checked_unref(obj) {
  gpointer __objref;

  g_object_add_weak_pointer((gpointer)obj,&__objref);
  g_object_unref((gpointer)obj);
  fail_unless(__objref == NULL, NULL);
}
```

Thomas Wabner <waffel@users.sf.net>
Stefan Kost <ensonic@users.sf.net>

GObject Properties

- GObjects have introspectable properties
  - min, max and default values
- Test if implementation handles limits
- To Do: error handling

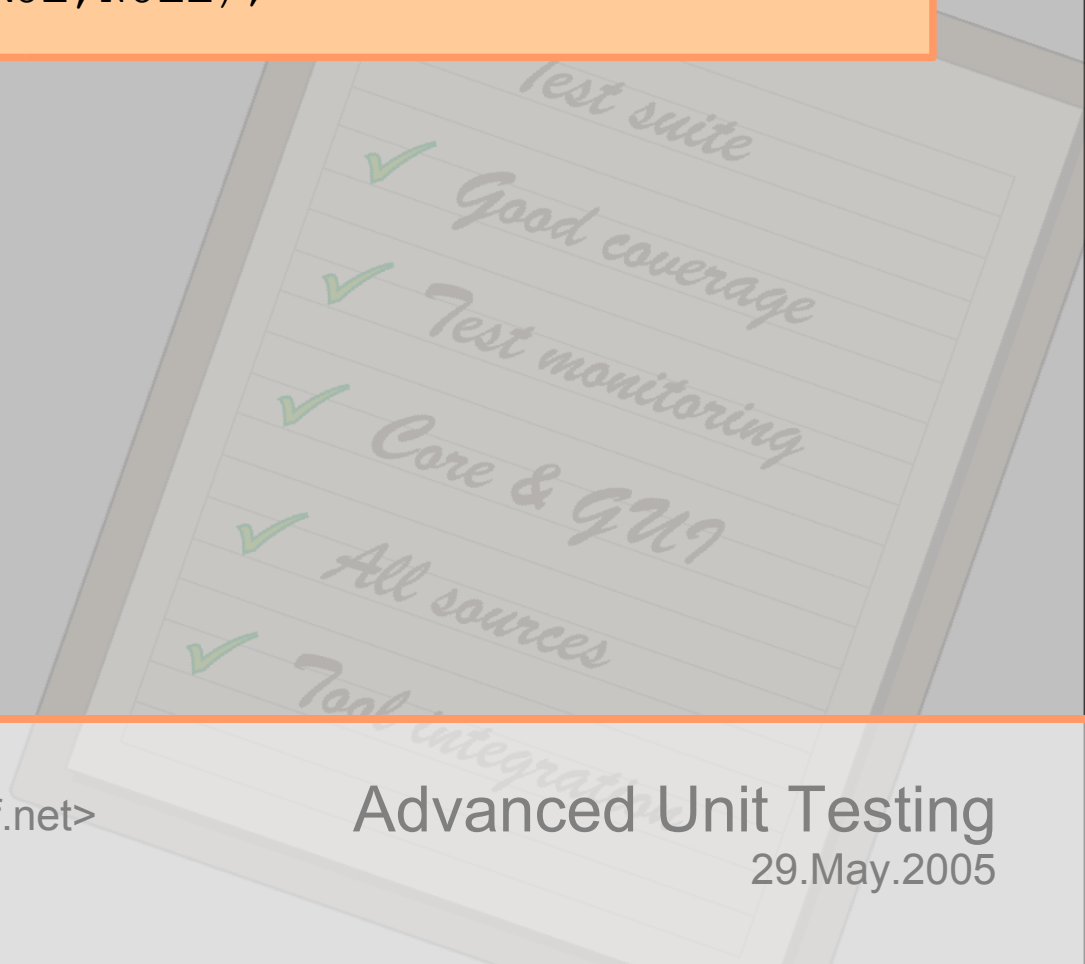


Thomas Wabner <waffel@users.sf.net>
Stefan Kost <ensonic@users.sf.net>

GObject Properties

Test:

```
gboolean check_prop_ret;

check_prop_ret=check_gobject_properties(obj);
fail_unless(check_prop_ret==TRUE,NULL);
```



Thomas Wabner <waffel@users.sf.net>
Stefan Kost <ensonic@users.sf.net>

Advanced techniques

- Supervised testing (Test monitoring)
  = using tools to see „what's going on“
- Locating bugs
- Presentation tests (UI)

Valgrind

- Tools to debug/profile x86-Linux programs
- Additional test target in Makefile.am
- runs all tests through valgrind again

http://www.valgrind.org

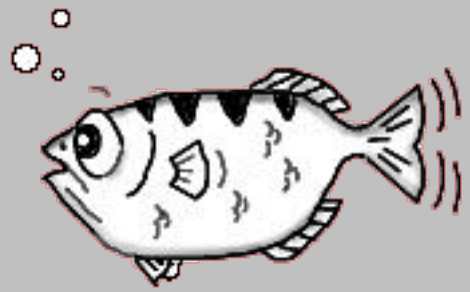
Valgrind

Test:

```
VALPREFIX = $(TESTS_ENVIRONMENT) GLIBCPP_FORCE_NEW=
VALDEFAULT = @VALGRIND@/bin/valgrind
VALSUPP = @VALGRIND@/lib/valgrind
VALSUPPRESSIONDEF = --suppressions=$(VALSUPP)/default.supp
VALSUPPRESSIONOWN = --suppressions=$(top_builddir)/buzztard.supp
VALSUPPRESSION = $(VALSUPPRESSIONDEF) $(VALSUPPRESSIONOWN)
VALOPTIONS = --tool=memcheck -q --trace-children=yes \
  --show-reachable=yes --leak-check=yes --num-callers=20
VALCMD = $(VALPREFIX) $(VALDEFAULT) $(VALOPTIONS) $(VALSUPPRESSION)

valgrind:: $(TESTS_BIN)
  for i in $^; do \
    rm -rf /tmp/$$i.valgrind.pid*; \
    $(VALCMD) --log-file=/tmp/$$i.valgrind ./$$i; \
  done
```

GNU Debugger : gdb

- Locate the source position of a bug
- Tests are uninstalled binaries
- Check runs tests as separate processes



http://www.gnu.org/software/gdb/gdb.html

GNU Debugger : gdb

Run uninstalled binary:

```
> cd tests
> libtool --mode=execute gdb ./bt_core
```

Do not fork tests:

```
> cd tests
> env CK_FORK="no" libtool --mode=execute gdb ./bt_core
```

Thomas Wabner <waffel@users.sf.net>
Stefan Kost <ensonic@users.sf.net>

Coverage Analysis

- How much of our code is covered by tests?
- Number of tests is not a good indicator
- gcov + lcov
  - gcov is part of gnu compiler collection
  - lcov is developed by the LTP (Linux Testing Project)

http://ltp.sourceforge.net/coverage/lcov.php

Coverage Analysis

Compile with coverage options:

```
CFLAGS="-fprofile-arcs -ftest-coverage"
```

Generate report:

```
mkdir ./coverage
lcov --directory . --zerocounters
make check

# workaround for lcov not liking libtool
for file in `find . -name "*.da" | grep ".libs" ` ;do
  mv $file `echo $file | sed -e 's/\/\.libs//'`;
done

lcov --directory . --capture --output-file ./coverage/bt.info
genhtml -o ./coverage --num-spaces 2 ./coverage/bt.info
```

Coverage Analysis

LTP GCOV extension - code coverage report

Current view: directory - lib/core
Test: buzztard.info
Date: 2005-03-19 Instrumented lines: 3282
Code covered: 57.2 % Executed lines: 1878

Filename	Coverage		
application.c		75.3 %	55 / 73 lines
core.c		90.9 %	10 / 11 lines
gconf-settings.c		47.0 %	31 / 66 lines
machine.c		67.6 %	346 / 512 lines
pattern.c		43.0 %	86 / 200 lines
plainfile-settings.c		0.0 %	0 / 55 lines
playline.c		82.8 %	96 / 116 lines
processor-machine.c		50.0 %	27 / 54 lines
sequence.c		41.4 %	108 / 261 lines
settings.c		88.1 %	37 / 42 lines
setup.c		72.5 %	177 / 244 lines
sink-machine.c		59.3 %	54 / 91 lines
song-info.c		46.2 %	54 / 117 lines
song-io-native.c		66.2 %	339 / 512 lines
song-io.c		71.3 %	92 / 129 lines
song.c		55.1 %	75 / 136 lines
source-machine.c		78.2 %	43 / 55 lines
timeline.c		39.0 %	46 / 118 lines
timelinetrack.c		61.3 %	46 / 75 lines
tools.c		26.3 %	5 / 19 lines
wave.c		0.0 %	0 / 67 lines
wavelevel.c		0.0 %	0 / 67 lines
wavetable.c		58.6 %	41 / 70 lines
wire.c		57.3 %	110 / 192 lines

Generated by: LTP GCOV extension version 1.4

```
101        7 : gboolean bt_song_play(const BtSong *self) {
102        7 :   gboolean res;
103          :
104        7 :   g_return_val_if_fail(BT_IS_SONG(self),FALSE);
105          :
106          :   // emit signal that we start playing
107        6 :   g_signal_emit(G_OBJECT(self), signals[PLAY_EVENT], 0);
108        6 :   res=bt_sequence_play(self->priv->sequence);
109          :   // emit signal that we have finished playing
110        6 :   g_signal_emit(G_OBJECT(self), signals[STOP_EVENT], 0);
111        6 :   return(res);
112          : }
113          :
114          : /**
115          :  * bt_song_stop:
116          :  * @self: the song that should be stopped
117          :  *
118          :  * Stops the playback of the specified song instance.
119          :  *
120          :  * Returns: %TRUE for success
121          :  */
122        0 : gboolean bt_song_stop(const BtSong *self) {
123        0 :   gboolean res;
124          :
125        0 :   g_return_val_if_fail(BT_IS_SONG(self),FALSE);
126          :
127        0 :   res=bt_sequence_stop(self->priv->sequence);
128        0 :   return(res);
```

Thomas Wabner <waffel@users.sf.net>
Stefan Kost <ensonic@users.sf.net>

GUI Tests

- How to avoid flickering windows ?
- Using a virtual X server

http://xorg.freedesktop.org/

Thomas Wabner <waffel@users.sf.net>
Stefan Kost <ensonic@users.sf.net>

GUI Tests

Virtual framebuffer X display:

```
Xvfb :9 -screen 9 1024x786x16
```

Using it:

```
./my-app  --display=:9
```

To do:

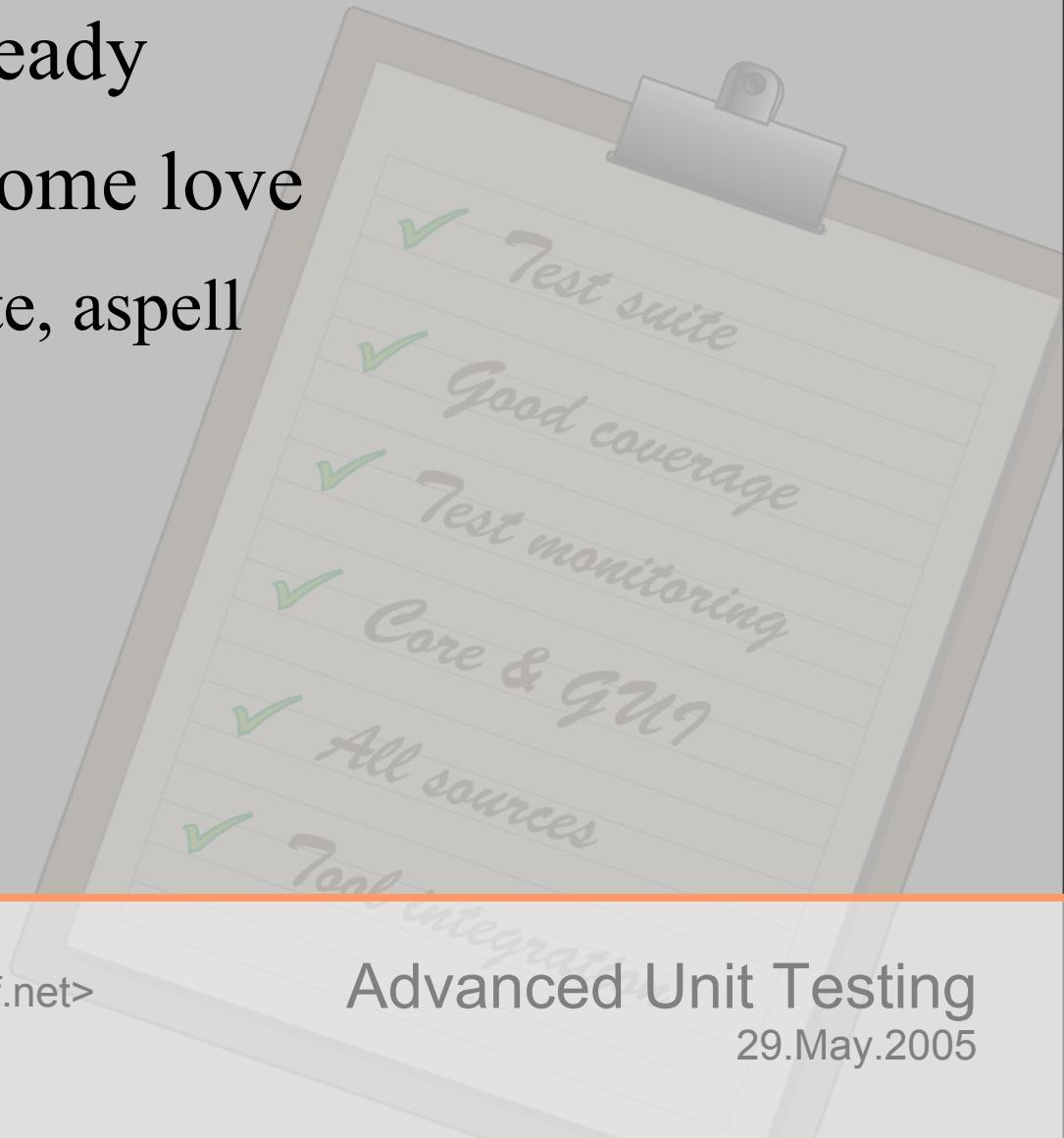
- Make tests using it automatically
  - Use an env-var (e.g. CK_DISPLAY)

Quality assurance

- Testing is more than Unit testing
  - XML files
  - Language catalogs
  - API docs completeness
  - Spelling of docs and catalogs
  - Coding style

Conclusion

- Become test infected!
- Much can be tested already
- Still many tools need some love
  - e.g. lcov, intltool-update, aspell
- Feedback wanted!



Questions

Thomas Wabner <waffel@users.sf.net>
Stefan Kost <ensonic@users.sf.net>