QuviQ QuickCheck

Tired of writing and maintaining thousands of automated tests? And did you know that repeating tests finds only 15% of your bugs anyway? Let QuickCheck generate new tests for you daily, saving you effort and nailing your bugs earlier!

Three steps to QuickCheck

- Write a *QuickCheck specification* instead of test cases—general properties your system should always satisfy.
- QuickCheck uses *controlled random generation* to test your code against the spec.
- Failing cases are *automatically simplified* before they are presented to you.

QuickCheck takes you quickly from specification to identified bug.

Concise Specifications

Your QuickCheck specifications consist of *properties* and *generators* which describe system behaviour in a specified set of cases. Each property can generate many different test cases—so specifications can be much more concise and maintainable than test suites. At the same time, many more cases can be generated, so testing is more thorough. QuickCheck uses the power of *functional programming* to keep specifications concise and readable.

Controlled Randomness

QuickCheck tests your properties in randomly generated cases, either interactively or invoked from your test server. Pure random generation makes for poor test data, but QuickCheck's simple and powerful interface puts *you* in control, making it easy to generate complex data with specific properties, with tight control over the distribution of test cases. You can build complex generators easily by composing simple ones. For example, if you define

```
tod() ->
    {choose(1,12),
     choose(0,59),
     oneof([am,pm])}.
```

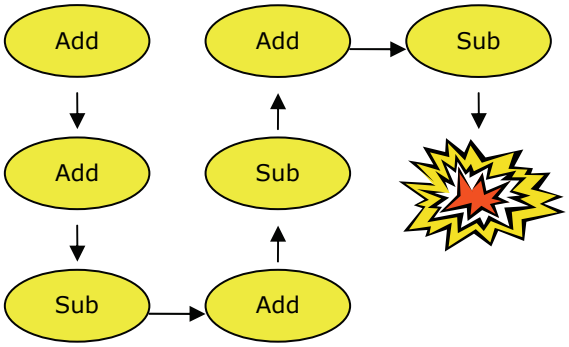
to generate times of day, then `list(tod())` generates sequences of different times—no more code than that is needed.

Automatic Simplification

Conventional random testing generates failing cases in which—like failures from the field—the *signal* causing the failure is obscured by a great deal of random *noise*, making fault diagnosis difficult and costly. QuickCheck automatically *simplifies* failing cases to minimal examples that provoke the failure, making fault diagnosis easy. Once again, *you* are in control: while useful simplification strategies are built-in, you can easily extend or replace them with your own.

State Machine Testing

It is often convenient to model the system under test via a *state machine*. QuickCheck's state machine library lets you model the state declaratively, write pre– and postconditions and state transition functions for each operation, then generate minimal call sequences that provoke violations. QuickCheck can exercise a wider variety of call sequences than you can possibly write manual test cases for. Testing a pre-release version of Ericsson's Media Proxy, with commands to *Add*, *Modify*, and *Subtract* callers, provoked numerous failures.



*This failing case for Ericsson's Media Proxy was simplified from a sequence of 160 commands. The cause was data corruption on the **first** Subtract. No shorter sequence provoked the same fault.*

Open Source

Quviq's specification language is not proprietary—it is Erlang, a concurrent, fault-tolerant functional language developed at Ericsson, with a high-quality compiler and extensive libraries developed and maintained as open-source by Ericsson. Only the QuickCheck libraries themselves are proprietary. Linking QuickCheck specifications to code under test in other languages (such as C), or via TCP/IP, is easy to do thanks to Erlang's already good support.

Need Help?

Sound interesting? Can't yet see how to deploy QuickCheck in your own situation? **Let us help you!** With our expertise from the research frontier, we can help you get the best out of QuickCheck in your particular application. Licences, training courses, and consultancy are available. Contact us at **`www.quviq.com`**.