Refactoring Functional Programs
Experience of building refactoring tools for Haskell and Erlang

Functional Programming

Functional programs are distinctive in emphasising values rather than variables, and in making functions first-class data. Modern functional languages also offer fine control over side-effects.

Haskell and Erlang

Functional languages differ considerably. Haskell is strongly typed, evaluated lazily, and has layout-sensitive syntax. Erlang is weakly typed, strict and layout-insensitive.

Haskell has a de jure standard, Haskell 98, and a de facto standard implementation, Glasgow Haskell. Open-Source Erlang has a controlled series of system releases.

Functional refactoring

Some refactorings have OO equivalents, e.g. generalisation

- Generalise printList over the print operation; this is wrapped in a fun, containing the side-effect.

Others address functional / concurrent aspects: e.g. convert asynchronous to synchronous communications.

Experience report

Cover the complete language of the standard ... including as much of the macro mechanism and pre-processor as possible. De facto standards more relevant than de jure.

Integrate with IDEs, and make sure these are those used in practice. If they are editors (emacs, gvim) also work with tools: makefiles and test frameworks.

Preserve program appearance: layout and comments crucial. Pretty printers do not always do this ... and even a ten line program can become unrecognisable.

Refactoring = Transformation + Condition: in practice it is much more difficult to implement the condition (e.g. preserve binding structure) than transform (e.g substitute names).

A refactoring will have many variants: e.g. operate on one / all / some occurrences when generalising. How to present these alternatives to the user?

Compensate or fail? If a condition fails, should the system compensate for this and perform the refactoring, or fail, and expect the user to make the compensation explicitly?

Infrastructure is needed to support the analysis ... use existing systems whenever possible.

User-defined refactorings: what is the right language? Should it be an API, or a domain-specific language?

Implementing a refactoring system which will be used by the practising programmer requires it to meet a variety of non-functional usability constraints as well as presenting choices and variants to the user.

HaRe refactorer embedded in the gvim editor

The tools - HaRe (Haskell) and Wrangler (Erlang) - include structural, naming, data type and module refactorings.

Simon Thompson
www.cs.kent.ac.uk/~sjt/
HaRe http://www.cs.kent.ac.uk/projects/refactor-fp/
Wrangler http://www.cs.kent.ac.uk/projects/forse/wrangler/doc/