Required

# Code Editing

* Pervasive syntax highlighting.
  + More than just s-expressions and typical values: regex literals, [interpolated strings](http://cemerick.com/2009/12/04/string-interpolation-in-clojure/), etc.
* Brace, bracket, paren matching.
* paredit equivalency.
* Easily-togglable rainbow parens (I usually hate them, but then end up thinking they’d be handy for ~5 seconds).
* s-expression navigation.
* all of the above needs to deeply configurable – there are lots of different preferences out there, and I hold at least half of them from time to time!
* code completion
  + Should have visibility to locally-defined vars/bindings, those that are :required or :used, as well as stuff in core
  + Host (Java) interop completion should be key off of the type hints in the surrounding code, and do what it can with imports and such
  + In all cases, provide associated docs as well as parameter hints (that are bolded, etc. as you go along adding parameters)
* in-place popup macroexpansion

# Project and Dependencies Management

* maven (and eventually, clojure [polyglot maven](http://polyglot.sonatype.org/)) support
  + ensure that project settings, REPL configuration, etc. etc. are all driven from the POM
* insofar as lein becomes a mainstay of clojure development, support will be needed for it as well
* “Zero-config”
  + All clojure projects (whether maven- or lein-based) have clojure as a dependency, so requiring any special “clojure setup” or creation of a “clojure platform” is either pointless or dangerous (the latter if you are unwittingly using a statically-defined “clojure platform” while your project’s POM is explicitly declaring a different version of clojure as a dependency) [2](http://cemerick.com/ideal-clojure-development-environment/#fn2)
  + The one area where having a default set of Clojure libraries available (as a “platform”, if it must be called that) is to start a REPL for a non-Clojure project, which **is** very convenient.

# Code Analysis

* Refactoring (rarely needed, but when needed, it would be **incredibly** helpful)
  + This would almost certainly be limited in static usage – local renames, and such.
  + Given an open REPL that’s had my entire application loaded (and therefore should be able to use function/var metadata to its fullest), I should be able to do damn near anything you can do in a top-notch Java IDE (within the bounds of what’s relevant, anyway).
  + the equivalent of Java IDEs’ “organize imports” in clojure ns use/import/require clauses
* Static analysis
  + Symbol navigation
    - Go to declaration
    - A “Go to Var” option, corollary to “Go to File” or “Open Resource”; I should be able to type “redu” to find and go to reductions, or type “w-o” to find and to go with-open, in any namespace in any open project
  + Find usages
    - Making this work across projects would be absolutely stellar.  i.e. If I’ve got two projects open, I should be able to find usages in project B of a var defined in a namespace in project A.
* current-file code outline
* static namespace browser, with reasonable search capabilities

# UI/UX

* Sane UI/UX
  + progressive disclosure, discoverability of functionality, familiar help systems, etc
  + Supporting, nay, embracing mousing. Yes, I use the mouse (actually, a trackball) in addition to the keyboard, get over it. Much of my time spent “programming” is actually spent doing things other than typing (thank goodness), and many of these tasks are not best done with the keyboard.
  + All of this **should** go without saying IMO, but emacs seems to put a stake in the ground that much of the above is unnecessary or undesirable. The issue of progressive disclosure and discovery of functionality is a particular sore spot for me. Many of the most important features of a development environment (or perhaps any class of nontrivial software) are the most esoteric that might go untouched for weeks or months (debuggers, profilers, code coverage tools, configuration settings, etc), and one must be able to **effectively** stumble through them for the first time, and after having not touched them for a long time. I don’t want to have to learn, remember, or google for the M-x run-clojure-debugger command or somesuch along with 200, 500, 1000? others.

# REPL

* REPL support (overall, I think it’d be hard to improve on enclojure[1](http://cemerick.com/ideal-clojure-development-environment/" \l "fn1)– and, AFAIK, swank-clojure, in this department)
* all REPLs should be separate processes (I don’t want a REPL session to hork my main environment)
  + therefore, all REPLs are fundamentally remote REPLs
  + ensure that the REPL server library is lean, available, stable, and licensed liberally, so I can roll it into my applications and therefore….
  + connect to running REPL servers on any IP/port from the dev environment, making all REPL-enabled functionality available regardless of where the host REPL process is running
* REPL history persistent across IDE restarts
* Support for multiple REPLs (i.e. load code into the last focused REPL)
* Full editor capability in the REPL – highlighting, symbol completion, formatting, etc.
* Browsable/searchable REPL history
  + Prior expressions accessible via standard command history (Ctrl-up/down, etc), as well as by clicking on expressions still shown in the REPL window
* Runtime namespace browser, with the same usage characteristics as the static one, but tied to the focused REPL
* Ability to check status of and kill long-running REPL invocations
* Optional (likely default) separation of input, \*out\*, and \*err\* content
* Automatic generation and configuration of the classpath for local REPLs; this includes (almost certainly in this order!):
  + all current-project source paths (this ensures that changes source files are loaded before same-named files or AOT’ed classfiles from the project’s artifact(s))
  + all **other** projects’ source paths (this allows me to load changes I’ve made to code across my “main” project’s dependencies)
  + all project dependencies (no-brainer)
  + the project’s artifact(s) (or perhaps target/classes (in the case of maven projects) is enough – obviously necessary in order to have access to classfiles from other languages/tools that generate them)

Nice to have

# Code Editing

* automatic structural formatting
  + don’t apply the same rules to docstrings as would be applied to other multiline string literals

# Java Interop

* absolutely top-notch Java development support
  + this one item almost necessitates that I’ll always be in a major IDE, i.e. Eclipse, IntellJ, or NetBeans. A lot of the lower-level plumbing that’s necessary to pull off things like a complete debugger, profiler, etc. etc. is so complex, I can’t imagine anyone matching “the big three” without simply co-opting one of them.
* debugger and profiler that are able to effortlessly “focus” on clojure-level or java-level code/data structures depending on what I’m interested in at the time

# Code generation

* generation of deftype/defrecord/extend-type/gen-class/proxy method scaffolding
  + Roughly the equivalent of the “implement method” quickfix that is available in Java IDEs when you add a superclass or interface to a class definition; i.e. be able to type(proxy [java.awt.event.MouseAdapter], and have shells of the various methods from the specified interface and/or abstract superclasses dropped into place. The same thing should be applicable to all forms that generate classes from clojure.
* code coverage tools that hook into clojure.test and, again, can track clojure- and/or java-level coverage equally well

# REPL

* Configurable pretty-printing of output
  + This also means being able to “print” non-textual data, such as having images drawn inline into the REPL if so desired, etc.
  + See [Factor’s REPL](http://re-factor.blogspot.com/2010/09/visual-repl.html) for an example