

Just For Fun

Adam Keys

RubyConf 2009

<http://therealadam.com>

Inspiration

Tuesday, December 8, 2009

I wrote the proposal for this talk right after Why the Lucky Stiff disappeared himself. I'd taken a step back and was thinking about what I'd learned from Why. The thing that stood out the most to me was that he didn't seem to be too concerned with the seriousness of his code. He was not in it to show how professional he was, how robust his code was, or to espouse his ideas as superior to all others.

Since Why inspired me in doing this talk, I thought I'd highlight some of the projects I found most embodied his aesthetic.

Why ☺ Potion

```
PN potion_def_method(Potion *P, PN closure, PN self, PN key, PN method) {
    int ret;
    PN cl;
    struct PNVtable *vt = (struct PNVtable *)self;
    unsigned k = kh_put(id, vt->kh, ((struct PNString *)key)->id, &ret);
    if (!PN_IS_CLOSURE(method)) {
        if (PN_IS_PROTO(method))
            cl = potion_closure_new(P, (PN_F)potion_proto_method, PN_NIL, 1);
        else
            cl = potion_closure_new(P, (PN_F)potion_getter_method, PN_NIL, 1);
        PN_CLOSURE(cl)->data[0] = method;
        method = cl;
    }
    kh_value(vt->kh, k) = method;
#ifdef JIT_MCACHE
#define X86(ins) *asmb = (u8)(ins); asmb++
#define X86I(pn) *((unsigned int *)asmb) = (unsigned int)(pn); asmb += sizeof(unsigned int)
#define X86N(pn) *((PN *)asmb) = (PN)(pn); asmb += sizeof(PN)
    {
        u8 *asmb = (u8 *)vt->mcache;
        #if __WORDSIZE != 64
            X86(0x55); // push %ebp
            X86(0x89); X86(0xE5); // mov %esp %ebp
            X86(0x8B); X86(0x55); X86(0x08); // mov 0x8(%ebp) %edx
        #define X86C(op32, op64) op32
        #else
            #define X86C(op32, op64) op64
        #endif
        #endif
        for (k = kh_end(vt->kh); k > kh_begin(vt->kh); k--) {
            if (kh_exist(vt->kh, k - 1)) {
                X86(0x81); X86(X86C(0xFA, 0xFF));
                X86I(kh_key(vt->kh, k - 1)); // cmp NAME %edi
                X86(0x75); X86(X86C(8, 11)); // jne +11
                X86(0x48); X86(0xB8); X86N(kh_value(vt->kh, k - 1)); // mov CL %rax
            }
            #if __WORDSIZE != 64
                X86(0x5D);
            #endif
            X86(0xC3); // retq
        }
        X86(0xB8); X86I(0); // mov NIL %eax
        #if __WORDSIZE != 64
            X86(0x5D);
        #endif
        X86(0xC3); // retq
    }
    #endif
    return method;
}
```

Tuesday, December 8, 2009

First off is Potion. It's a nifty little language that is influenced by Lua in its implementation and a ton of other languages syntactically.

I was a little surprised that Why decided to hack out his own language. I'd previously thought it was something fairly dry. There's a reason that there's a website where you have to separate pictures of serial killers from language designers. They're usually, present company excepted, cut from a very different cloth. But Why showed me how one could do language stuff and do it playfully.

Why ☺ Camping

```
%w[uri stringio rack].map{|l|require l};class Object;def meta_def m,&b
(class<<self;self end).send:define_method,m,&b end end;module Camping;C=self
S=IO.read(__FILE__)rescue nil;P=<h1>Cam\ping Problem!</h1><h2>%s</h2>
U=Rack::Utils;Apps=[];class H<Hash
def method_missing m,*a;m.to_s=~/$/?self[$]=a[0]:a==[]?self[m.to_s]:super end
undef id,type;end;module Helpers;def R c,*g
p,h=/^(.+?)\//,g.grep(Hash);g-=h;raise"bad route"unless u=c.urls.find{|x|
break x if x.scan(p).size==g.size&&/^#{x}\/?$/=~(x=g.inject(x){|x,a|
x.sub p,U.escape((a[a.class.primary_key]rescue a))}}
h.any?? u+"?">U.build_query(h[0]):u end;def / p
p[0]==?/?@root+p:p end;def URL c='/*a;c=R(c,*a) if c.respond_to?:urls
c=self/c;c=@request.url[/.{8,}?(?=\//)+c if c[0]==?;/URI c end
end;module Base;attr_accessor:input,:cookies,:headers,:body,:status,:root
def render v,*a,&b;mab(/^_/!~v.to_s){send(v,*a,&b)} end
def mab l=nil,&b;m=Mab.new({},self);s=m.capture(&b)
s=m.capture{layout{s}} if l && m.respond_to?(:layout);s end
def r s,b,h={};b,h=h,b if Hash==b;@status=s;
@headers.merge!(h);@body=b;end;def redirect *a;r 302,"','Location'=>URL(*a).
to_s;end;def r404 p;P%#{p} not found"end;def r500 k,m,e;raise e;end
def r501 m;P%#{m.upcase} not implemented"end;def to_a
@env['rack.session']=@state;r=Rack::Response.new(@body,@status,@headers)
@cookies.each{|k,v|next if @old_cookies[k]==v;v={value=>v,path=>self/"'"} if
String===v;r.set_cookie(k,v)}
r.to_a;end;def initialize(env,m)
r=@request=Rack::Request.new(@env=env)
@root,@input,@cookies,@state,@headers,@status,@method=
r.script_name.sub(/\/$/,''),n(r.params),
H[@old_cookies = r.cookies],H[r.session],{m=~/r(\d+)/?$1.to_i: 200,m
end;def n h;Hash==h ?h.inject(H[]){|m,(k,v)|m[k]=n(v);m}: h end;def service *a
r=catch(:halt){send(@method,*a)};@body||=r
self;end;end;module Controllers;@r=[];class<<self;def r;@r end;def R *u;r=@r
Class.new{meta_def(:urls){u};meta_def(:inherited){|x|r<<x}}end
def D p,m;p='/'if !p||!p[0]
r.map{|k|k.urls.map{|x|return(k.instance_method(m)rescue nil)?
[k,m,*$~[1..-1]][:I,'r501',m]if p~/^#{x}\/?$/}};[:I,'r404',p] end
N=H.new{|_,x|x.downcase}.merge! "N"=>'(\d+)',"X"=>'([^\//]+)',"Index"=>'
def M;def M;end;constants.map{|c|k=const_get(c)
k.send:include,C,Base,Helpers,Models;@r=[k]+r if r-[k]==r
k.meta_def(:urls){["/#{c.scan(/.[^A-Z]*/).map(&N.method(:[]))*'/'}"]
}if !k.respond_to?:urls}end end;I=R()
end;X=Controllers;class<<self;def goes m
Apps<<eval(S.gsub(/Camping/,m.to_s),TOPLEVEL_BINDING) end;def call e
X.M;p=e['PATH_INFO']=U.unescape(e['PATH_INFO'])
k,m,*a=X.D p,e['REQUEST_METHOD'].downcase
k.new(e,m).service(*a).to_a;rescue;r500(:I,k,m,$!,:env=>e).to_a;end
def method_missing m,c,*a;X.M;h=Hash==a[-1]?a.pop: {}
e=H[Rack::MockRequest.env_for('',h[:env]||{}))
k=X.const_get(c).new(e,m.to_s);k.send("input=",h[:input])if h[:input]
k.service(*a);end;def use*a;m=a.shift.new(method(:call),*a);meta_def(:call){|e|
m.call(e)}end end;module Views;include X,Helpers end;module Models
autoload:Base,'camping/ar';def Y;self;end end;autoload:Mab,'camping/mab';C end
```

Tuesday, December 8, 2009

Finally, there's Camping. Following the development of Camping for a little while, I got a glimpse into Why's aesthetic. He explicitly didn't want Camping to be a Rails competitor. He just wanted it to be Camping. And to fit on a t-shirt.

```

LINK_TEXT_WITH_TITLE_RE = /
  ([^"]+?)      # $text
  \s?
  \(((^)+?)\)\  # $title
  $
/x
def link(opts)
  if opts[:name] =~ LINK_TEXT_WITH_TITLE_RE
    md = LINK_TEXT_WITH_TITLE_RE.match(opts[:name])
    opts[:name] = md[1]
    opts[:title] = md[2]
  end
  "<a href=\"#{escape_attribute opts[:href]}\#{pba(opts)}>#{opts[:name]}</a>"
end

```

Why ☺ RedCloth, hpricot, syck

```

void
syck_parser_add_level( SyckParser *p, int len, enum syck_level_status status )
{
  ASSERT( p != NULL );
  if ( p->lvl_idx + 1 > p->lvl_capa )
  {
    p->lvl_capa += ALLOC_CT;
    S_REALLOC_N( p->levels, SyckLevel, p->lvl_capa );
  }

  ASSERT( len > p->levels[p->lvl_idx-1].spaces );
  p->levels[p->lvl_idx].spaces = len;
  p->levels[p->lvl_idx].ncount = 0;
  p->levels[p->lvl_idx].domain = syck_strndup( p->levels[p->lvl_idx-1].domain, strlen( p->levels[p->lvl_idx-1].domain ) );
  p->levels[p->lvl_idx].status = status;
  p->lvl_idx += 1;
}

```

Tuesday, December 8, 2009

Why's brought us plenty of other things too. Redcloth, hpricot, and syck. The latter is of particular note. I'm willing to bet very few of us are eager to write a YAML parser. And yet, Why did and now most of us use it as part of MRI. So clearly, you don't have to do fun things to have fun. You can do things that lower the friction towards others having fun.

Why 😊 **Craziness**

Tuesday, December 8, 2009

A survey of Why's works is incomplete without mentioning his mixed media works. His presentations at various conferences were amazing and if you've missed any of them, seek it out. The Poignant Guide is a touch-stone of our community. Though you're likely to either love it or find it a little wacky, no one can discount the influence Chunky Bacon and the cartoon foxes had in making our community one where a tongue placed firmly in cheek is welcome.

Why 😊 **Legacy**

Tuesday, December 8, 2009

Before Why disappeared his works, he had lamented about the temporary nature of software. It's true that hardly does the dust settle on the current hotness before the new hotness supplants it.

Consider the notion of “building one to throw away” from The Mythical Man Month. In software, we're used to the notion that writing code is a learning process. It's not uncommon that we'll find we can do a better job from a clean slate.

But, what Why has showed us is that software can outlive its useful life. Code can exist as something that makes you happy and as something that makes others happy. Even if its only in the act of reading it, not all software ceases to be useful after people are done executing it.

Why 😊 **Thanks, Why.**

Q: Is the Ruby community still fun?

Tuesday, December 8, 2009

Back to the matter at hand.

As I was thinking of this proposal, I was struggling with my own work. More accurately, I was struggling with my lack of `_finished_` work. It had been months since I felt like I'd `_made_` something of consequence. Plenty of big ideas, but no finish. Through the lens of Why's work, I came to look at things from a different angle.

I started to think, maybe we've lost track of fun here in the Ruby community. I've observed a lot of consternation about whether Ruby's release management is good enough, whether it's GC is suitably comparable, or whether we're all just a bunch of weenies distracted by shiny new....oh look!

My point on this talk `_was_` going to be that we need to get back to having fun. Rekindle that amazement we first felt when we typed some stuff in and made the computer do something `_different_`.

A: Yes!

Tuesday, December 8, 2009

But it turns out, people are totally having fun. They're doing games. They're doing graphics. They're doing sound. They're doing _real things_ in the _real world_.

People out there are doing graphics. You've got Ruby Processing, a way to use JRuby to write Processing sketches. You've got Ruby on Acid, a way to create loopy graphics. If you're willing to go further a-field, now is a great time for doing graphics on the web. Processing-JS, Raphael, Protovis and just about everything else built on Canvas or SVG are seeing wider support.

Other folks are out there doing sound. Giles Bowkett is making probabilistic music. People are hacking things up with OSC. [More].

People ☺ Polymathin'

```
class ConnectionDelegate

  def initialize(parent, &block)
    @parent = parent
    @block = block
  end

  def connectionDidFinishLoading(connection)
    doc = NSXMLDocument.alloc.initWithData(@receivedData,
                                           options:NSXMLDocumentValidate,
                                           error:nil)

    if doc
      @block.call(doc)
    else
      @block.call("Invalid response")
    end
  end

  def connection(connection, didReceiveResponse:response)
    case response.statusCode
    when 401
      @block.call("Invalid username and password")
    when (400..500)
      @block.call("Unable to complete your request")
    end
  end

  def connection(connection, didReceiveData:data)
    @receivedData ||= NSMutableData.new
    @receivedData.appendData(data)
  end

  def connection(conn, didFailWithError:error)
    @parent.status_label.stringValue = "Error communicating with Twitter"
  end
end
```

Tuesday, December 8, 2009

One of my heroes lately is Greg Borenstein. He brought us a very clever way to write Ruby and run it on an Arduino. After a stint working on a music site, Grabb.it, he's now at NYU's ITP program. He's bringing together his talent for music, hardware hacking, software and design to make cool stuff. Sounds like fun.

People ☺ **Study groups**

```
fib :: Int -> Integer
fib =
  let _fib 0 = 1
      _fib 1 = 1
      _fib x = fib (x - 2) + fib (x - 1)
  in (map _fib [0..] !!)

fibSeq :: Int -> [Integer]
fibSeq len = map fib [1..len]

-- Actually, this should run fib until the result is greater than 4M, but
-- just playing around with the numbers was slightly easier. Also, I
-- understand compose now.
solve :: Int -> Integer
solve max = sum . filter (< 4000000) . filter even $ fibSeq max
```

Tuesday, December 8, 2009

Jim Weirich and a bunch of other people are doing study groups based around the Wizard book. They get together in person or on a mailing list and talk through solutions to the problems. Sometimes they even solve them in other languages! If number problems are more your thing, Project Euler has a bunch of problem sets that are fun to work through, but also great for using as motivation to teach yourself a new language.

```

static OBJ TrVM_defmethod(VM, TrFrame *f, OBJ name, TrBlock *b, int meta, OBJ receiver) {
    TrFunc *func;
    if (b->arg_splat)
        func = (TrFunc *) TrVM_interpret_method_with_splat;
    else if (kv_size(b->defaults) > 0)
        func = (TrFunc *) TrVM_interpret_method_with_defaults;
    else
        func = (TrFunc *) TrVM_interpret_method;
    OBJ method = TrMethod_new(vm, func, (OBJ)b, -1);
    if (meta)
        TrObject_add_singleton_method(vm, receiver, name, method);
    else
        TrModule_add_method(vm, f->class, name, method);
    return TR_NIL;
}

```

People 😊 Language hacking for profit fun

```

AsgnCall = ( rcv:Value '.'
             )? ( rmsg:Message '.'
                 ) * msg:ID - asg:ASSIGN
                - val:Stmt
           { rcv = 0 }
           { rcv = NODE2(SEND, rcv, rmsg) }
           { VM = yyvm; $$ = NODE2(SEND, rcv, NODE2(MSG, SYMCAT(msg, asg), NODES(NODE(ARG, val)))) }

Receiver = (
             | rcv:Call
             | rcv:Value
             )
           { rcv = 0 }
           { $$ = rcv }

```

Tuesday, December 8, 2009

Marc Andre-Cournoyer, the guy who wrote Thin, is having fun hacking on languages. If you check out his GitHub page, you'll see a plethora of little language experiments he's done. He even self-published a book on how to join in on the language hacking fun. Charles Nutter has been going in a very similar direction with his own little languages.

Fun, srsly

Tuesday, December 8, 2009

Further, there are people having fun doing things many of us don't consider fun. Folks are doing excellent work on virtual machines, asynchronous IO frameworks, and improving the XML consumption landscape. Their work enables us to have more fun.

```

CallInst* VMLLVMMethod::call_operation(Opcode* op, Value* task,
                                       Value* js, BasicBlock* block) {
    const char* name = InstructionSequence::get_instruction_name(op->op);
    Function* func = operations->getFunction(std::string(name));
    if(!func) {
        std::string str = std::string("Unable to find: ");
        str += name;

        throw std::runtime_error(str);
    }

    std::vector<Value*> args(0);
    args.push_back(task);
    args.push_back(js);

    switch(op->args) {
    case 2:
        args.push_back(ConstantInt::get(Type::Int32Ty, op->arg1));
        args.push_back(ConstantInt::get(Type::Int32Ty, op->arg2));
        break;
    case 1:
        args.push_back(ConstantInt::get(Type::Int32Ty, op->arg1));
        break;
    }

    return CallInst::Create(func, args.begin(), args.end(), "", block);
}

```

Fun 😊 Virtual machines

```

# Using an array of Implementation objects, +methods+, print a switch
# statement which decodes the arguments and calls the function that
# contains the implementation of the instruction.
#
def generate_decoder_switch(methods, io, flow=false)
  io.puts "switch(op) {"

  methods.each do |impl|
    io.puts "  case #{impl.name.bytecode}: { // #{impl.name.opcode}"

    args = impl.args
    case args.size
    when 2
      io.puts "    int #{args[0]} = next_int;"
      io.puts "    int #{args[1]} = next_int;"
      io.puts "    #{impl.body}"
    when 1
      io.puts "    int #{args[0]} = next_int;"
      io.puts "    #{impl.body}"
    when 0
      io.puts "    #{impl.body}"
    end

    io.puts "  break;"
    io.puts "  }"
  end

  io.puts "default:"
  io.puts "%q! std::cout << \"Invalid instruction: \" << InstructionSequence::get_instruction_name(op) << \"\\n\"; abort();!"
  io.puts "}"
end

```

Tuesday, December 8, 2009

Let's start with virtual machines. Love 'em or hate 'em, folks like Koichi, Evan Phoenix, Brian Ford, Charles Nutter, Tom Enebo, Laurent Sansonetti and Brian Lam are out there busting their butts to make us hotter, faster Rubies. Worst of all, they're not even writing in Ruby! They're knee deep in C++, Java, C# or Objective-C.

[What a wonderful smell you've discovered.]

Now, to some folks, writing a virtual machine is fun. To write one for a language you love is even better. Now, it's probably the case that these folks don't exactly love Ruby anymore; I'm sure they'd all love to vanish a feature from the language to make our lives easier. But in toiling away on building better Rubies, something that is by no means easy, they are making it possible for all of us to have much more fun. These dudes are my heroes.

```

module Gen1HoodEdges
  class Mapper < Wukong::Streamer::Base
    def process rsrc, src, dest, *_
      src = src.to_i ; dest = dest.to_i
      yield [ src, '>', dest ]
      yield [ dest, '<', src ]
    end
  end

  class Reducer < Wukong::Streamer::AccumulatingReducer
    # clear the list of incoming paths
    def start! target, dir, *args
      print target + "\t" + dir # start line with target and list type
    end
    def accumulate target, dir, neighbor
      print "\t" + neighbor # append neighbor to output, same line
    end
    def finalize
      puts '' # start new line
    end
  end

  class Script < Wukong::Script
    def default_options
      super.merge :sort_fields => 1, :partition_fields => 1
    end
  end
end

```

Fun ☺ Tauntaun hackers

```

#--
# I know I'm missing something stupid, but the inside of class << s
# can't see locally-bound values. It can see globals, though.
def set_receiver blk
  $em_____tmpglobal = blk
  class << self
    define_method :call, $em_____tmpglobal.dup
  end
end

```

Tuesday, December 8, 2009

Next, let's talk about another interesting group of people. How many of you have heard that Ruby can't scale? If I had a nickel for every time I heard that Ruby will always lag behind due to its immature GC, it's naive IO or it's quaint concurrency model, I could buy myself a really nice scotch! And yet, there are folks hacking away in these areas, making it easier to push Ruby to its limits.

Joe Damato and [tmm1] are hacking around in the threading bits. Evan Weaver and [MBARI guy] are using their scanning electron microscopes to improve MRI's memory behavior and garbage collection. The Phusion guys are hacking out `_Apache modules_` to make running Ruby apps easier. [EM guy] is building EventMachine so that we can write apps that don't need to lean on threading so heavily. Philip Kromer is building tools so you can operate on Big Data using Ruby. Ilya Grigorik is writing tutorials so we can see how all these neat toys fit together. There's fun to be had, you just have to decide on what's fun; sometimes it's in the bowels of the code.

[Lineup of developers, pick the guys that can write an Apache module.]

Fun ☺ **Wookiee barbers**

```
module Nokogiri
  if self.is_2_6_16? && !defined?(I_KNOW_I_AM_USING_AN_OLD_AND_BUGGY_VERSION_OF_LIBXML2)
    warn <<-eom
    HI. You're using libxml2 version 2.6.16 which is over 4 years old and has
    plenty of bugs. We suggest that for maximum HTML/XML parsing pleasure, you
    upgrade your version of libxml2 and re-install nokogiri. If you like using
    libxml2 version 2.6.16, but don't like this warning, please define the constant
    I_KNOW_I_AM_USING_AN_OLD_AND_BUGGY_VERSION_OF_LIBXML2 before requiring nokogiri.
    eom
  end
end
```

Tuesday, December 8, 2009

Lastly, I'd like to single someone out as a great example. REXML is bundled with Ruby and raised the bar on what I'd consider a humane library for dealing with XML. It's a delightful API. But internally, it's a little pokey. I'd long contemplated sitting down with something like libxml and building a decent API. I knew it'd be a net awesome for the community.

But I didn't do it because, frankly, I want as little XML in my life as possible. Really, it's not so much the XML I dislike as the things that `_generate_` the XML; but I digress. I decided not to do this because I didn't want to make something and abandon it. But Aaron Patterson came along, did it, and has stuck with it. Now Ruby has an `_excellent_` library for dealing with XML. `_And_`, it has more Aaron Patterson. Win-win.

Fun 😊 **[This could be you]**

Tuesday, December 8, 2009

If you ever sit around thinking, "man I'd love to work on a virtual machine, system-level library, or handy but thankless application library, but that's not going to get me on Hacker News", think again. It might not get you on Hacker News, but who cares? If you can succeed in doing one of these things, you can have fun and at the same time enable other people to have fun.

Fun at RubyConf 2009

Tuesday, December 8, 2009

So, good news for you: there's plenty of fun to be had. You can even go to some talks about it in the next couple days!

Jeff Casimir is talking about generative art in this talk "Code of Art". Ron Evans and Damen Evans are talking about the flying Arduino robots they built in "Flying Robot". Ninh Bui is talking about game development in the aptly titled "Game Development with Ruby". Noah Thorp is talking about generating music in "Making Music with Ruby". Logan Barnett and Jay McGavern are talking about making games with JRuby in "Ruby Game Development with Jemini". Aaron Patterson and Ryan Davis are going to wow us with some fantastically bad ideas in the surprisingly titled "Worst. Ideas. Ever." Matt Aimonetti is going to show us how to build games with MacRuby in "Writing 2-D games for OS X with Ruby". Finally, Sarah Mei is showing us how to get kids interested in coding in "Indoctrinating the Next Generation".

So there's plenty of fun to be had. Hell, there's three presentations just on making games. You've got no excuse not to make it to at least one of these.

Fun  Enjoying  Creating  Contributing

Tuesday, December 8, 2009

But, bad news for me. I've still got to make up some problem and then show you how to solve it. And it's got to have some believable relation to the notion of having fun.

And thusly, I started thinking more about enjoyment. The goodness of the things we make. How to make more stuff.

I ended up on an interesting tangent. All of us have something we want to make. The process of making it and the end product is fun.

But, lots of us, myself as an exemplar, encounter road blocks to making it. We're either not making enough of it, we're not happy with how what we make affects our peers, or worse, we're not happy making it.

So at the risk of going down a tangent that is perhaps too "soft" for RubyConf, I'm going to share some solutions I've found for these problems. None of them are panacea. They're things to try when you get stuck. Things to do when you're not having fun.

If you got a problem...

Tuesday, December 8, 2009

There are three sorts of problems I'm going to tackle today. If you arrange them in a sort of "Hierarchy of Open Source Developer Needs" they move from having more fun and removing non-fun, making more things, and making more meaningful contributions to the community you're part of.

I haven't completely cracked this nut. I've got some ideas about things that help me along towards these goals. But I haven't found the silver bullets yet, if they exist at all. What I do have are a bunch of hacks, patterns, whatever you'd like to call it, that seem to help. When I find I'm not progressing at the rate I'd like to, I employ one or more of the following ideas to get myself back on track.

Enjoying

Tuesday, December 8, 2009

Enjoying **Do it for you**

Tuesday, December 8, 2009

Most of us write software for monies. This puts us in a habit of taking other people's needs and translating it into working code. When you do this in a particular domain long enough (say web apps or iPhone apps), you start to get a particular set of itches; you notice you're always coming up with user authentication systems or ways to write views. Soon enough, it can seem like those sorts of things are really interesting to you.

While it's possible those things are interesting to you, it's more likely that your brain has sort of convinced you of this so that you keep on working. For a select few, writing just-in-time compilers or wrappers for XML parsers is a good time. Praise be to those sorts of people, because they make all our developer lives easier.

But for the rest of us, it helps to step back and ask what kind of project we're really interested in. Maybe we'd rather write something that does interesting things with music or graphics. Maybe making a game would be more fun. If you're like me, you've got all sorts of ideas bouncing around in your head that are basically joke applications.

Deciding to devote time to one of these projects is fun because you matter. As much as possible, you should toil for your own reasons, not just because the money is great. Many of us are lucky to work with the tools and languages we enjoy and get paid to do it. But it can go further than that: work on the problems you enjoy with the people you enjoy.

Do it for you.

TODO:

- * Write down all the projects you've wanted to tackle but convinced yourself were too "impractical" or not sufficiently "adult". Reconsider putting some time towards them.
- * Think about the work-for-money you do on a daily basis. Do you enjoy it? If so, why is that? If not, what could you change to make it more enjoyable? How can you change your

Enjoying **Don't burn out**

Tuesday, December 8, 2009

We all love thinking about the awesome things that will happen once we've shipped our project. Large bags of money, huge recognition by our peers, and unicorns surely await us so soon as we ship, publish, or launch. So we put our noses to the grindstone and work our tails off until we get there.

Working hard is great. Most people want to work hard at something, and if you can get outsized rewards for doing so, great! But, there's a flip-side to keep in mind.

Burn-out.

It's easy to get so deep into something that it consumes you. Your well-being becomes intertwined with it. If the project is going well, you're in a great mood and things are splendid. If it's going poorly, you're feeling down, cranky and unmotivated.

Therefore, it's really important to pace yourself. If you find that you're just going through the motions, not making progress, or dreading the work, stop! Take a step back. Figure out if you've taken a wrong turn somewhere. Get away from the project for a spell, just to clear your head.

Our bodies are clever things. When I started working out, my muscles would ache for a couple days. At first, running was pretty painful. But as I got into it, they started to hurt less. Eventually, I got to the point where if a muscle was hurting, I knew I was doing some motion wrong.

Our minds seem to work in a similar way. My brain feels sharp and focused on days that I do my best work. It feels dull when I'm going down the wrong path or am too frustrated to work on something.

Sometimes it seems like our culture demands of us to push through. Just muscle up and get

Enjoying Exercise

Tuesday, December 8, 2009

If you're like me, you read that headline and groaned. Exercise is boring, tedious and necessarily hard work. Isn't it easier to read a book, play a video game or watch a movie? Yes.

However, after I started a real workout program, strength training with a trainer and doing cardiovascular workouts, I noticed my brain was sharper. My workouts had the same mind clearing and focusing effect that walks had, but to a larger extent. The morning after I went on my first significant run, my brain felt like it was in overdrive. Awesome ideas were coming left and right, my mind was focused, much of my procrastinating nature seemed to melt away.

Much of what we know about the human body, and in particular fitness, seems to change like programming language fashion. But there is some research that shows that exercise gets the brain and body working in ways that aid creative problem solving, like programming.

Further, I've found that exercise is a great way to wind down the day. After a long day of interacting with people, immersing myself in problems, and writing code, I look forward to my workout in the evening. It gives my brain a time to process the day's events. I can step away from the challenges of the day and figure out what's important and how to act on it.

Luckily, exercise programs are easy to do. You pick something that sounds interesting, try it for a few days and commit to it. If it's not interesting, pick something else.

Committing to an exercise program is a lot like committing to a project or goal. It's all about showing up. If you get to the gym or lace up your shoes enough, you'll make visible progress. If you sit down and grind on your project enough, you'll have something to show for it.

The hard part about projects and exercise is the commitment. It's a lot easier to want the results than it is to get them. But if you can commit to a project, you can commit to exercise (and vice versa).

Creating

Tuesday, December 8, 2009

Creating **Go to your thinking place**

Tuesday, December 8, 2009

Some days, I sit down at the computer, put my hands on the keyboard and...nothing. I bounce around between apps in the task switcher, I might surf the web. But, despite my excitement about the notion of whatever my project is, nothing related to it moves from my brain to my fingers to the screen.

We all find energy in the oddest places. Each of our brains is activated by different things. For mine, it's a hot shower or a leisurely walk. Yours might turn on after the first cup of coffee or a set of push-ups. The key is to try lots of different things until you've got a good idea what flips your brain into the "on" position.

These places and activities are what I call my "thinking" place. It's where I go to let the ideas bouncing around in my head come to the surface. Sometimes I go in with a problem that needs solving, but I'm not making progress sitting in front of a computer. Regardless, it's where I "go" to stew in ideas, ponder actions and outcomes, and figure out what to do next.

The great thing about a thinking place is that, often, your ideas smack you out of nowhere. You go in not expecting to think about some problem, let alone come up with a solution. But out it comes, obvious and well-formed. I know a good "thinking place" experience when I feel the burning need to jot all the ideas out on a whiteboard before they're forced out of my head with some other pressing or more practical need.

Another way to know you've found your thinking place is if it's a place you can talk to yourself and not feel absurd. This is why I like thinking on walks or in the shower. They are private activities where no one is around to wonder aloud if you're a little crazy or not.

TODO:

* Next time you're faced with a gnarly problem, get out from behind the desk. Go somewhere else and talk yourself through the problem.

Creating **Just start**

Tuesday, December 8, 2009

Procrastination is our enemy. It guides us away from what we want to do and towards the things we somewhat enjoy, but often wish we did less. If you want to read more about the insidious force that is procrastination, I highly recommend checking out [_The War of Art_](#). It's a short but highly effective read.

But for our purposes, let's talk about procrastination's greatest enemy: getting started. [The other enemy is sticking with it.] Overcoming inertia and putting the metaphorical pen to paper is often the hardest part. Once you give yourself permission to choose a tiny little corner of your problem and go with it, things get a lot easier.

Once you've got a little bit of the project outlined, it becomes much easier to pick part of it and start to flesh it out. This is, in my opinion, one of the reasons Rails appealed to so many developers early on. Instead of spending time picking ORMs, templates, or controllers, you run a few generator scripts and start filling in the blanks.

Giving yourself permission can route around analysis paralysis as well. Rather than think about all the pesky details or challenges you may come upon in your project, you come upon them as they really exist, rather than as they exist in a theoretical world inside your head. The real problems of any endeavor are often very different than those we predict inside our heads; addressing the real ones saves a lot of time.

The best thing about starting is that, well, you've started. Maybe you timebox the start so you don't chase false paths. Perhaps you just get one item on a list of ten things done. Sometimes it happens you can solve the whole problem in one sitting (while rare, these feel pretty great.) No matter how you tackle it, once you're done, you can look at what you've done and say "well, that's a start!" As you put it aside to work on other things, you can decide whether this project is worth continuing or whether it's better put aside as an interesting data point.

No matter the outcome, just starting is the important part.

Creating **Read code**

Tuesday, December 8, 2009

Another idea of how to keep the ball moving when your energy level is low is reading code. Just like a musician learns by listening and a writer learns by reading, developers can learn a lot and get new ideas by absorbing the code of others.

In your community, there are probably a few pieces of iconic software. Start by skimming through the sources of those. They may not be the best code; in fact they could be pretty shoddy internally, despite their utility. Reading code is a good way to start developing opinions and aesthetics on this topic.

I also like to read the code of the people I am inspired by. If I think what someone says on their weblog or presentations makes a lot of sense, I find it intriguing to dive into their code and see how their approach to problems differs and matches my own.

Reading code is also handy in that it can flip switches in your head. You may come across a solution to a problem you were thinking about earlier. It could inspire you to approach a problem from a new perspective. Or maybe it just motivates you to get back to it your project. No matter what, you win.

Code reading is easy. All you need is what you've already got. Download some sources, pick an example or test and start figuring out how the application or library works. Think about how you might make it better. Look at the shape of the project and contemplate how that makes you feel as a developer.

The nice thing about reading code is that you don't need to produce anything to say you're done. The end goal is to expand your mind. As long as you've done that, you're well off.

TODO:

* The next time you come upon an interesting project, download it immediately and read

Creating **Reconsider your schedule**

Tuesday, December 8, 2009

Making things requires time. Nothing is more frustrating than waking up with the enthusiasm to make something, but no time in which to harness that energy. So, it's worthwhile to rejigger your schedule so that you've got time to make the things that are important to you.

Most of us don't think too much about our schedules. We get up in time to prepare for work or school, go to school, eat a couple times, come home to unwind and then hit the sack in time to sleep well before it's time to rinse and repeat the next day. This works perfectly fine for a lot of people. But if you want to make awesome things, you need a more detailed schedule.

You've got to fit your time for creating in there somewhere. There's no fixed formula for this. It largely depends on knowing what your day looks like and what times of the day you are at your most energetic. Do you get stronger as the day goes on? Do you start off high and trail off throughout the day? This is the most important thing to know. You'll want to schedule the time you make things for yourself when you're at peak energy.

From there, most other things fall out. If you're a "morning person", you'll probably need to get up a couple hours earlier so you can make stuff before you go to work/school. If you're an afternoon person, you'll have to find a way to avoid friends, TV, video games, kids, etc. during your time for creation. If you're a night person, you have to figure out how to wrap your relaxation or family time up so you can get to making cool stuff.

Finding the time to make your awesome stuff is the easy part. Making the time is another ball of wax. Exercise, eating, bathing, books, movies, TV, sports, video games, socializing, family time, chores, and a myriad of other things tug on us. Somehow, we have to find a way to eliminate, minimize or mitigate the time we spend doing those things so we can make the awesome.

The easiest thing is to appraise how much enjoyment you get out of all the things you do that

Contributing

Tuesday, December 8, 2009

Contributing **Owe it to someone**

Tuesday, December 8, 2009

Many projects are too big for one person to tackle by themselves. Some need a diversity of skills not often present in one person. Others need a balance of opinions to keep decisions moving down the right path. Others just need more people toiling to take advantage of timing.

No matter why you end up working with someone else on a project, there's a handy side-effect of committing to working with someone towards a common goal: you end up owing them something (assuming you respect that person).

Some days you might wake up not particularly wanting to put in the hours on the project. Or perhaps there's a more interesting but less pressing part you'd like to work on today. Working by yourself, you may be tempted to skip out on the project or do the fun part. But if you owe someone, your personal pride and admiration for the other person can help nudge you in the direction of putting in the hours on the less exciting part.

Working with someone who you respect a lot is a powerful motivator for sticking with a project. But there's another benefit: it's exciting to work with people you deeply respect. Otherwise dull boring projects can become more interesting just from the interaction of working with awesome people.

The flip-side of owing it to someone is that they owe it to you. There will be days that you are uninspired by what you've accomplished lately, but they come up with work that amazes or energizes you. The benefit flows both ways.

When you're talking with exciting people, keep an ear open for projects you could work on together. Owing it to them could make an otherwise tough project far easier. Plus, you get to say you've worked with someone whose work you respect, and that is a great multiplier when you look back at what you've done lately.

Contributing **Take one for the team**

Tuesday, December 8, 2009

There are some kinds of code that are nasty. It is hard for me to look at ethernet drivers, email handling or code that deals with XML schema and think "this looks fun, I'd like to spend much of my spare time working on it!" But, the beauty of open source is that we all find fun in different things. Someone out there does think those things sound fun and will spend their time working on it.

If you put your economic philosopher hat on, you'd probably come to the conclusion that these people are really crucial to the the progression of our communities and our craft. Through their hard work, we can devote less of our brains to uninteresting detail work and more effort to things that are actually fun. Whether these folks are writing virtual machines, wrappers for solid XML parsers, or a way to modeling data easier, they deserve an outsized portion of our praise as a community.

What if you are one of these people, working on something you find rewarding, but the majority of the community around you finds tedious? There are a couple special rules that apply to maximizing your happiness while working on this sort of thing. First off, make sure people know about what you're doing; there's no sense toiling on the tedious fringes if people don't know you're there. Second, make it easy for people to make small, focused contributions; if they find a bug you're unlikely to come upon, removing friction to reporting an issue or patch will make your life easier. Third, and perhaps most importantly, make it easy for people to use your work. Read about "The Selfish Class" [reference] and make it simpler to use your work than starting their own or sticking with the status quo.

TODO:

* Review your favorite libraries with an eye towards which ones solve the most boring problems. Email the developer of those libraries to encourage them.

* Consider a problem space you find intriguing but lacking a good solution in your community. If it's something you can sustain an effort towards improving, get to it!

Contributing **Write words instead of codes**

Tuesday, December 8, 2009

Our minds are full of ideas. Sometimes they just pop out. Other times, they need a little help coming out. One way to help them out is to switch modes. Instead of trying to think something through, we can talk it through; the Prags call this "rubber ducky pairing".

Another way we can cajole our ideas is through writing. The act of writing utilizes different parts of our brain than coding does. We formulate ideas differently. We use different structures.

If you find yourself stuck, try switching from coding to writing. I've often found that by "talking to myself" through writing, I can better understand the problem at hand. If I can't get myself started on some code, sometimes I'll write the documentation or README first. After all, if you can't describe it, you probably shouldn't write it.

You can even take this a step further. Push the keyboard to the side and actually `_write_`. Use a pen. On paper. Take advantage of paper's random-access nature and write all over the page, instead of in a one dimensional stream of characters.

Or perhaps, you find yourself in a low energy situation. You'd like to make progress on your project, but you don't have the drive to jump into the code. So, write some documentation. Think about a tutorial for developers or a guide for users that would help the people you want to reach.

There are two tricks at play here. First, you're activating a different part of your brain. If you've been in a rut, maybe the solution is lurking in that part of the brain. If you're down on energy, perhaps this will serve as a running start. Secondly, it keeps you going. Even though you're not writing code, you're still making progress. While it might only be motion and not useful action, it is progress nonetheless.

TODO:

Make more awesome

Tuesday, December 8, 2009

I came at this talk with the idea that seriousness in the enemy. Always writing production-ready code and dismissing ideas that aren't sufficiently professional were my enemies at the time.

But over the months, I've come to regard a different kind of seriousness as the crux of the biscuit. Taking your own time seriously and deciding how you want to use it is important stuff.

Once I started to think about how I use my time, about what is important and fun versus what is necessary or extraneous, everything else fell into place. I thought about what was fun, examined the spectrum of how people are having fun doing those things and have started down the road of joining those people.

My hope is that by cataloging the kinds of fun you could have, showing specific ways specific people are having fun and then giving you some tools for choosing, creating, and contributing more fun, you'll come away from this weekend both re-energized and prepared to make more awesome stuff.

Thank you.

<http://therealadam.com>