# Design and Implementation of XMonad

A Tiling Window Manager

Don Stewart — dons@galois.com

Spencer Janssen — sjanssen@cse.unl.edu

## XMONAD

### A tiling window manager for X

- ➜ Automates window placement
- ➜ Tiles windows across screen to maximise screen use
- ➜ Mouse is optional
- ➜ Written, configured and extensible in Haskell
- ➜ Full multi-display/Xinerama support
- ➜ Fast, small, stable
- ➜ Active dev and user community
- ➜ Cool logo

Goal: productivity!

## We've been kicking around this idea forever

```
03.10.27:08:04:35
    <shapr> where would I start with a Haskell window manager?
04.05.23:20:35:27
    <platypus> is there a window manager written with haskell ?
04.06.02:10:05:31
    <shaleh> thinking about maybe a haskell based window manager
04.08.30:07:18:57
    * phubuh is investigating writing a window manager with hsx11
05.04.27:14:33:50
    <shapr> So, any haskell window managers?
05.12.31:02:01:06
    <twb> is there an X window manager written in haskell?
06.09.07:14:48:21
    <Deformative-II> So would haskell make a good window manager?
06.10.11:23:57:00
    <lispy> Smokey`: oh, write a purely haskell window manager
06.12.29:23:28:25
    <dylan> xcb + haskell = 100 line window manager? :)
```

## LET'S DO THIS!

About time we did something about this.

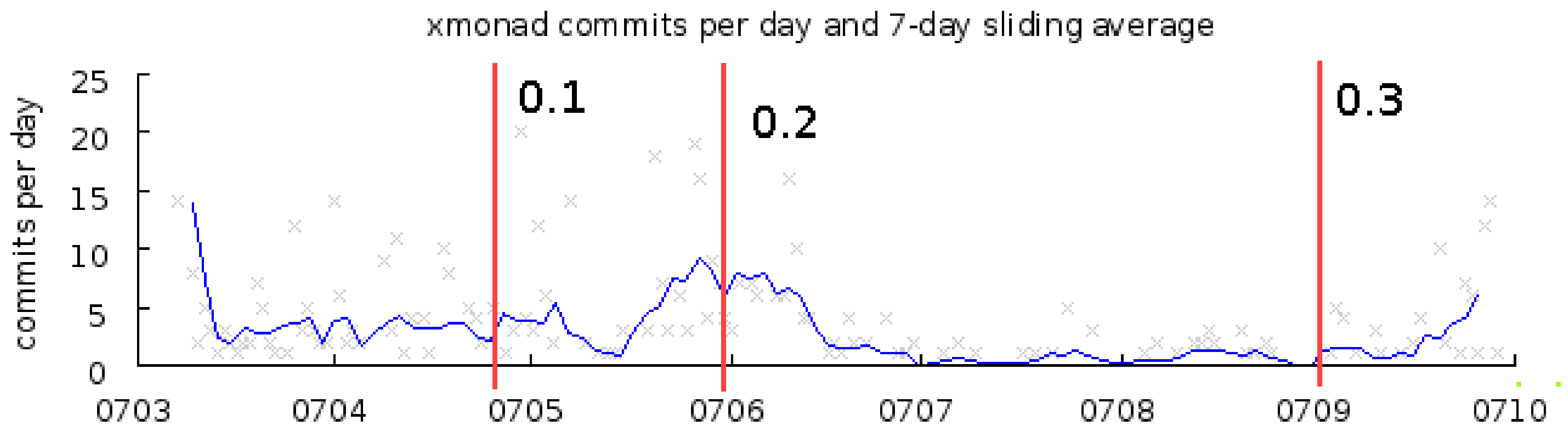
```
07.02.08:18:40:35
    <sjanssen> dons: I've started hacking on a window manager
07.03.06:19:52:10
    <dons> so i'm sitting here in the haskell window manager right now.
07.03.11:03:35:20
    <dons> so.. xmonad is now officially 'usable', imo (for laptops)
07.03.20:07:05:05
    * sjanssen is running a window manager in Haskell!
    With tiling!  So exciting!

07.06.01:20:02:14 <shapr> Yay xmonad!
```

## Comparison to similar projects

| | code | comments | language |
|---|---|---|---|
| metacity | $\gg$ 50k | | C |
| ion | 20k | 7k | C |
| ratpoison | 13k | | C |
| larswm | 6k | 1.3k | C |
| wmii | 6k | 1k | C |
| dwm | 1.5k | 0.2k | C |
| xmonad 0.2 | 0.5k | 0.7k | Haskell |

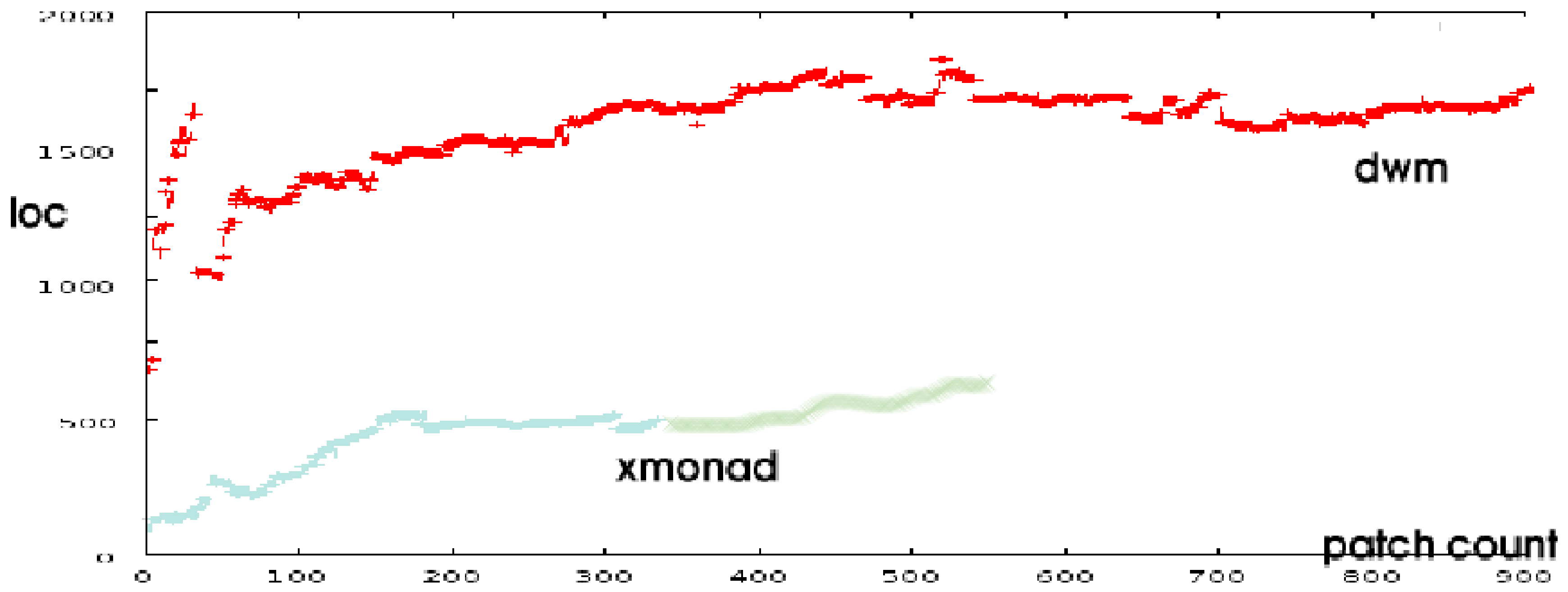
Seems reasonably typical result for a Haskell project.

## HISTORY OF THE PROJECT



Reliability is a guiding principle: makes Haskell look good

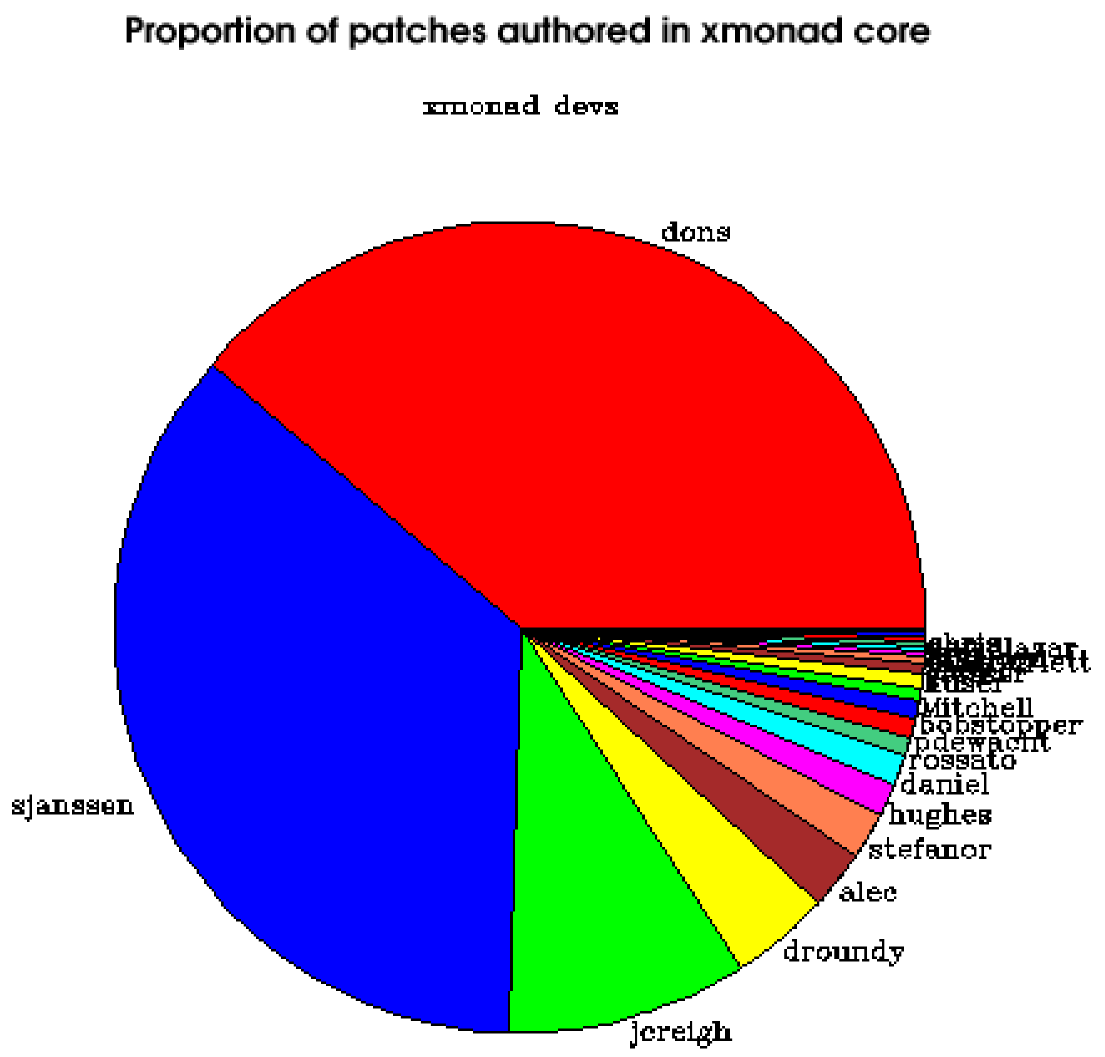
## Keeping Fit



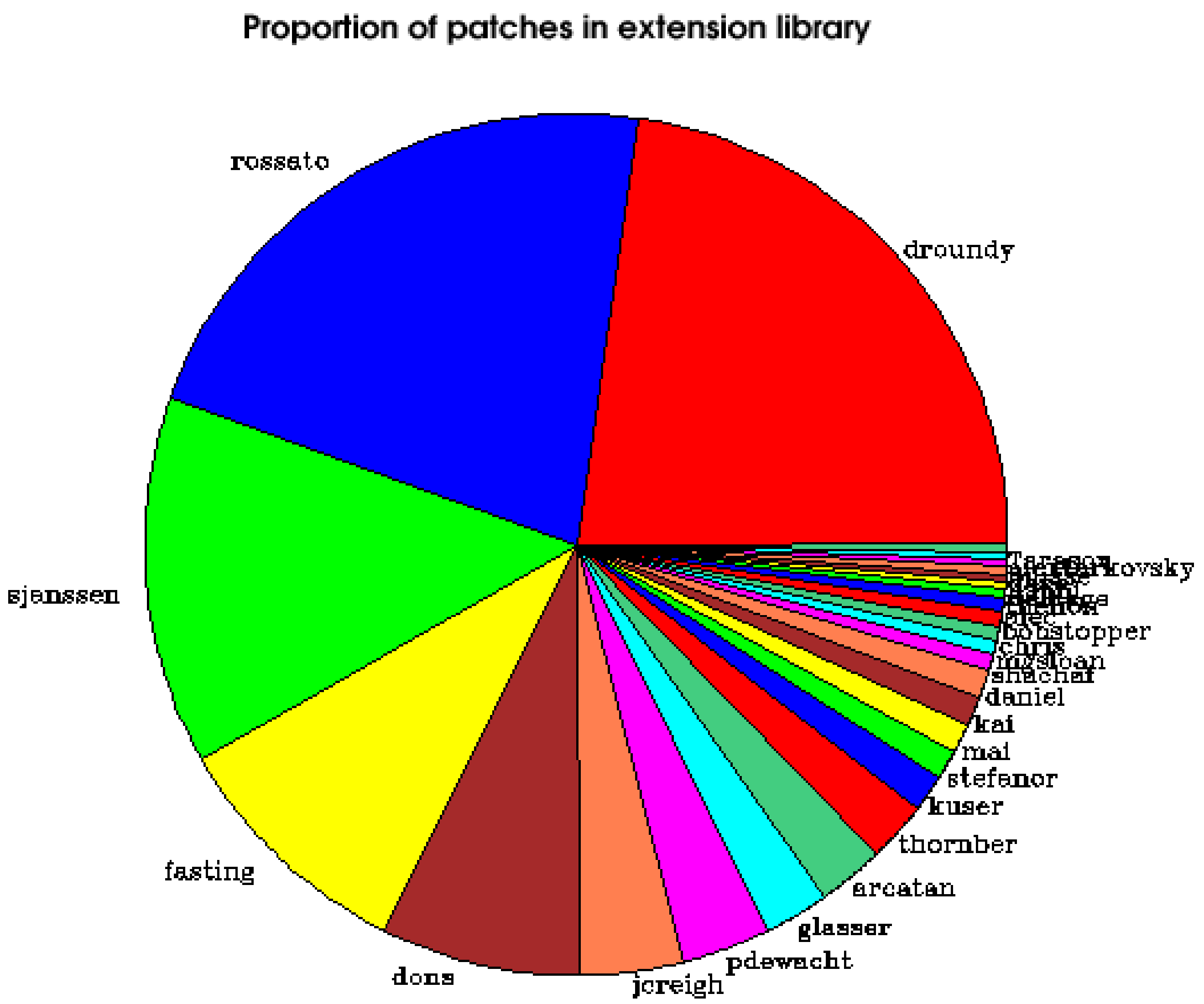
Refactor!

Lines of code tracking a useful heuristic:

- ➜ Code smell: bloat probably means something is wrong
- ➜ When to start refactoring

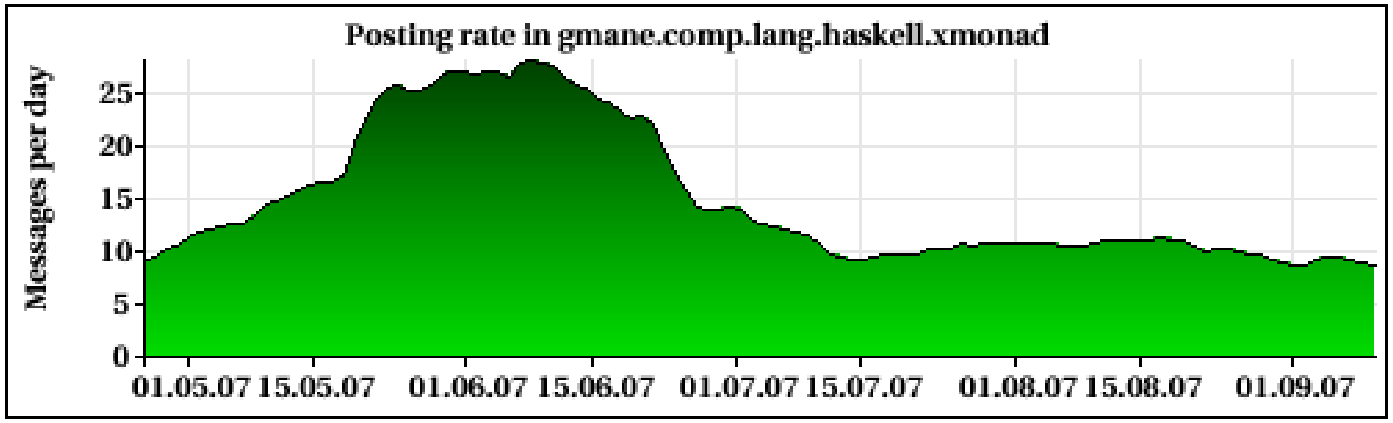


A distributed, online core team



Much larger extension hacking community: unexpected, but critical!

## ACTIVE COMMUNITY

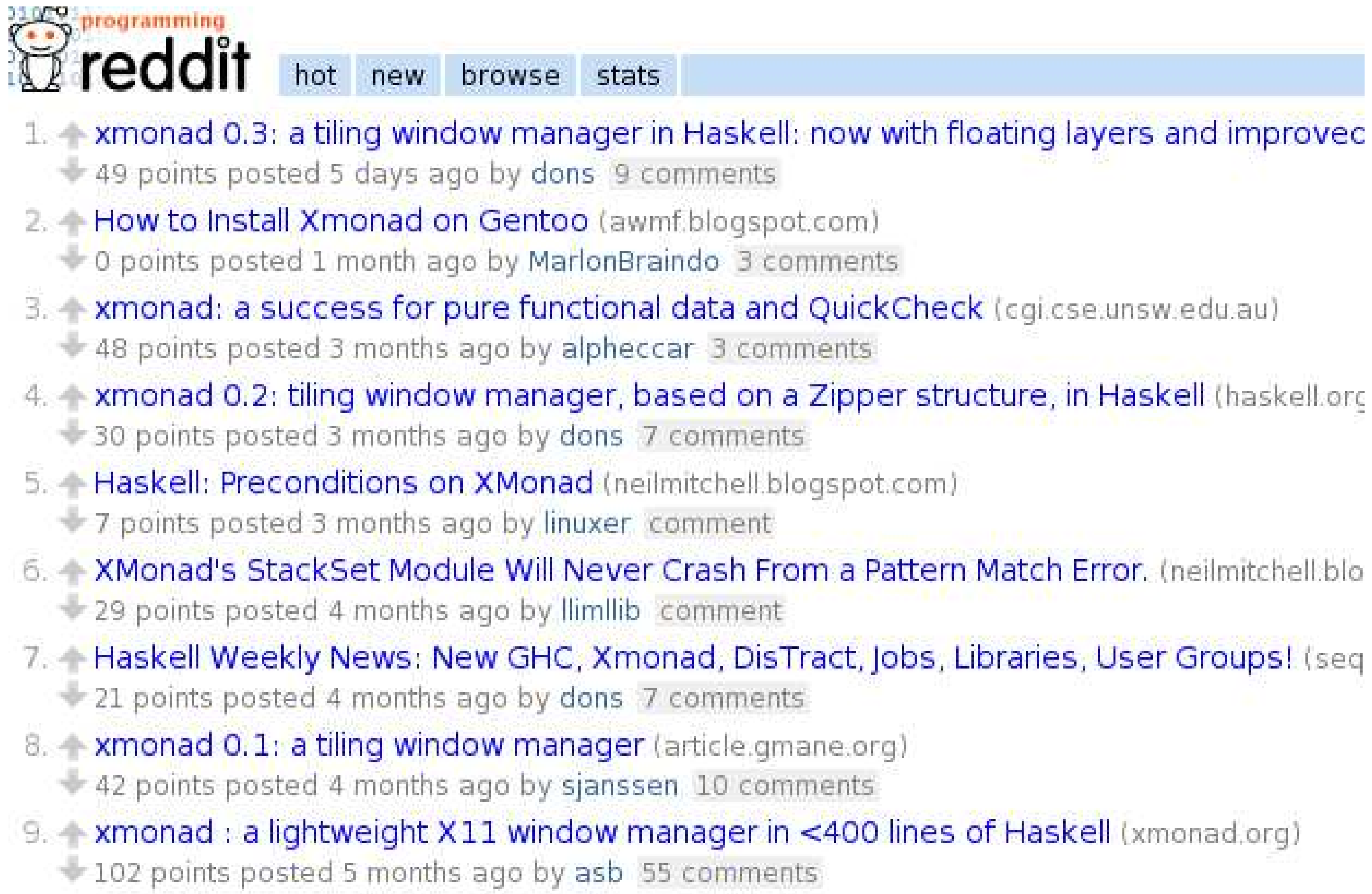


Around 150 on the mailing list, 70 on the IRC channel.

Developers and users freely intermix. Users become developers.

Explicit effort to build a friendly culture, as non-Haskellers arrive

## First Haskell project to really use the blogosphere?



1. xmonad 0.3: a tiling window manager in Haskell: now with floating layers and improvec
   49 points posted 5 days ago by dons 9 comments
2. How to Install Xmonad on Gentoo (awmf.blogspot.com)
   0 points posted 1 month ago by MarlonBraindo 3 comments
3. xmonad: a success for pure functional data and QuickCheck (cgi.cse.unsw.edu.au)
   48 points posted 3 months ago by alpheccar 3 comments
4. xmonad 0.2: tiling window manager, based on a Zipper structure, in Haskell (haskell.orç
   30 points posted 3 months ago by dons 7 comments
5. Haskell: Preconditions on XMonad (neilmitchell.blogspot.com)
   7 points posted 3 months ago by linuxer comment
6. XMonad's StackSet Module Will Never Crash From a Pattern Match Error. (neilmitchell.blo
   29 points posted 4 months ago by llimllib comment
7. Haskell Weekly News: New GHC, Xmonad, DisTract, Jobs, Libraries, User Groups! (seq
   21 points posted 4 months ago by dons 7 comments
8. xmonad 0.1: a tiling window manager (article.gmane.org)
   42 points posted 4 months ago by sjanssen 10 comments
9. xmonad : a lightweight X11 window manager in <400 lines of Haskell (xmonad.org)
   102 points posted 5 months ago by asb 55 comments

## We have fans.. and not all of them in this room!

"xmonad fits right into how I think window managers should be"

"xmonad is easily the fastest and has the smallest memory footprint I have found yet. "

"Suspiciously I relate to any software written in the "exotic" languages of programming. Usually either break is obtained or memory gorges much. But here everything is written on the fashionable nowadays Haskell, very rapid and memory it does not gorge." (Russian)

Break down stereotypes!

## WRITING A WINDOW MANAGER IN HASKELL

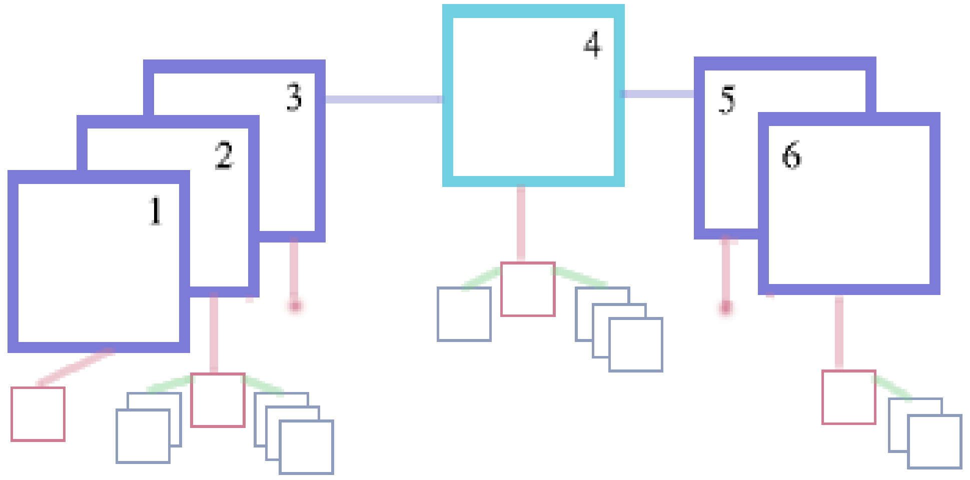
Pure models of impure systems

## The pure model: a zipper of window stacks

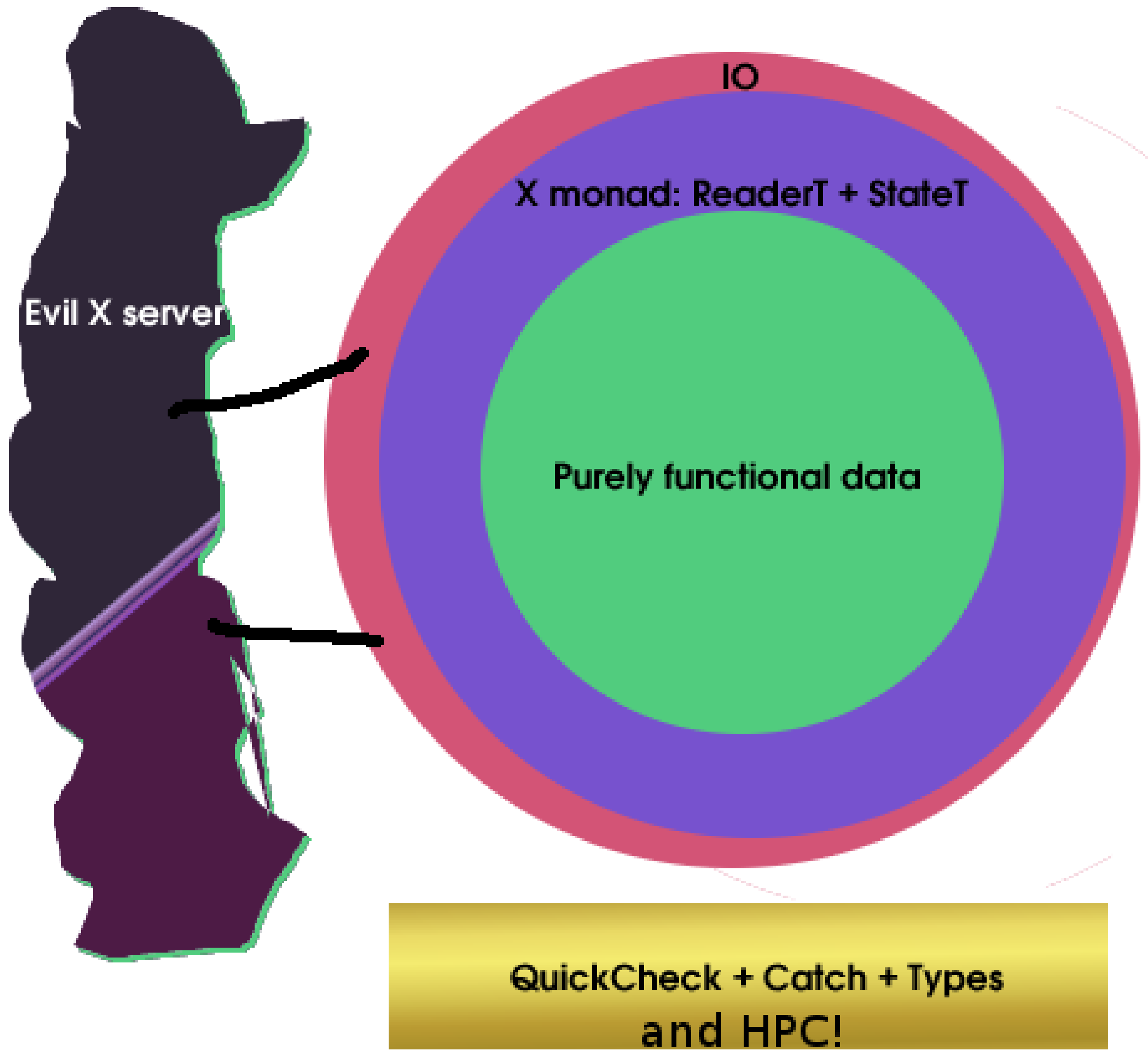
Step 1: Model your effectful program with purely functional data

Step 2: Find a natural api for it with QuickCheck

Step 3: Compile it and profit



- ➜ A cursor into a list of workspaces (Huet's zipper for lists)
- ➜ Each workspace is a cursor into a list of windows
- ➜ Small, orthogonal set of operations on this structure


Evil X server
IO
X monad: ReaderT + StateT
Purely functional data
QuickCheck + Catch + Types
and HPC!

## STACKSET

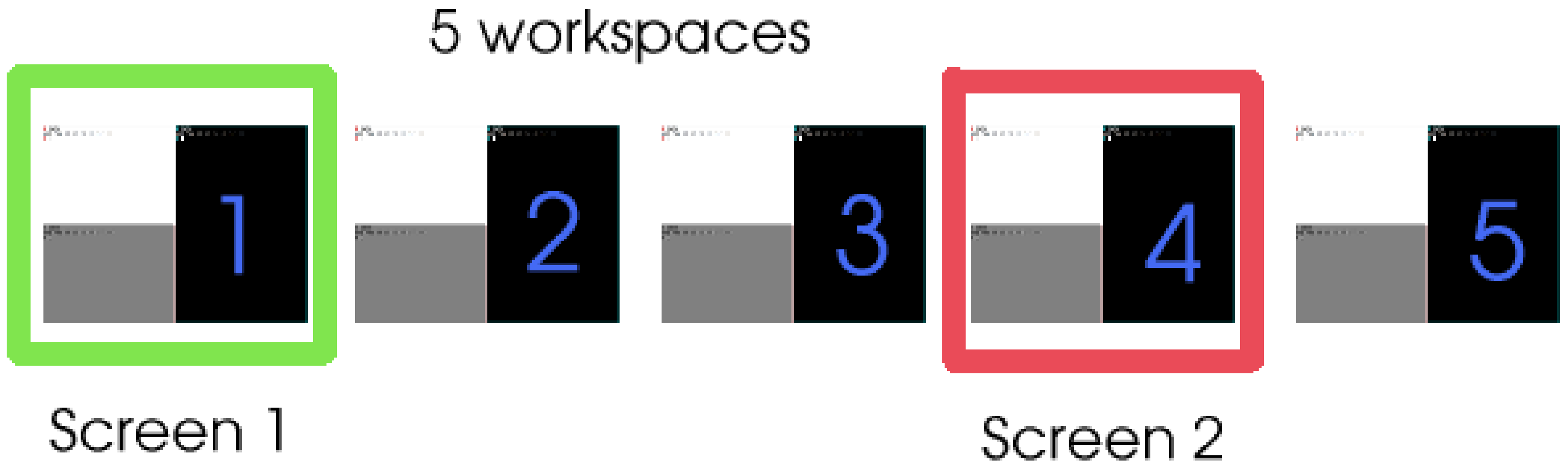
The core of xmonad is a simple window manager model

```
data StackSet i a =
     StackSet { current  :: !(Screen i a)
              , visible  :: [Screen i a]
              , hidden   :: [Workspace i a]      }

data Workspace i a = Workspace { tag :: !i
                                 , stack :: Maybe (Stack a) }

data Stack a = Stack { focus  :: !a
                     , up     :: [a]
                     , down   :: [a] }
```

The right structure makes the code smaller, faster and safer

Keep all logic in polymorphic, pure code! Don’t be tempted by the outside world.

## Mapping workspaces to multiple screens



Obvious mapping to multi-display support. Unusual for tiling window managers!

Workspaces either ‘current’, ‘visible’ or ‘hidden’.

## A Window Manager API

```
new    :: Int -> StackSet a
peek   :: StackSet a -> Maybe a
index  :: StackSet a -> [a]
insert :: a -> StackSet a -> StackSet a
delete :: a -> StackSet a -> StackSet a

focusLeft, focusRight :: StackSet a -> StackSet a
```

QuickCheck provided tool support to find the “axiomatic” window manager API: primitives were easier to identify when they had good QC properties

## MODIFYING A STACKSET: SHIFTING FOCUS

Shift focus one window ‘up’ the stack (just move the cursor):

```
focusUp :: Stack a -> Stack a

focusUp (Stack t (l:ls) rs)
        = Stack l ls  (t:rs)

focusUp (Stack t [] rs)
        = Stack x xs [] where (x:xs) = reverse (t:rs)
```

Guaranteed non-empty in the type.

Simplicity of implementation often corresponded to simpler semantics, and a more intuitive user interface.

## THE TYPE OF INTERACTIVE APPLICATIONS

xmonad has the usual type for a stateful, interaction app:

```
newtype X a = X (ReaderT XConf
                   (StateT StackSet
                       IO) a)


    deriving (Functor, Monad, MonadIO
             ,MonadState XState
             ,MonadReader XConf)
```

Carries around the window manager state and configuration values

Strong separation of X server calls from internal stuff.

*Key logic in pure, QuickCheck Haskell, confirmed with HPC*

## MODIFYING THE REAL WORLD

Pure model-view-controller stuff: receive events, updated a model, render back to the X server:

```
focusUp, swapUp :: X ()
focusUp    = windows W.focusUp
swapUp     = windows W.swapUp

shift, view :: WorkspaceId -> X ()
shift      = windows . W.shift
view       = windows . W.greedyView

windows :: (WindowSet -> WindowSet) -> X ()
```

`windows` is our thin monadic skin, rendering the model in X

## CONFIGURATION

Use a first class language for first class configuration

## CONFIG.HS: THE HASKELL TROJAN HORSE

We decided to be fundamentalists: configuration files will be Haskell only.

```
modMask :: KeyMask
modMask = mod1Mask

borderWidth :: Dimension
borderWidth = 1

defaultLayouts :: [Layout Window]
defaultLayouts = [ full, tiled, mirror tiled ]
```

This was a big win: when you can bind window manager events to user-written Haskell code, you jump way up the power spectrum.

Type safe, expressive config files!

## Emergent extension community

- ➜ People started sending each other config files.
- ➜ ... and asking for their favourite hack to be included
- ➜ Core devs grumbled
- ➜ And stuck them all in a darcs repo in the corner
- ➜ Then Andrea haddock-ised them
- ➜ And we published them on xmonad.org..

We didn’t know what we were getting in to...

```
$ cd XMonadContrib
$ ls
Accordion.hs             DynamicWorkspaces.hs
Mosaic.hs                SshPrompt.hs
Anneal.hs                Dzen.hs
NamedWindows.hs          Submap.hs
Circle.hs                FindEmptyWorkspace.hs
NoBorders.hs             SwitchTrans.hs
Combo.hs                 FlexibleManipulate.hs
README                   Tabbed.hs
Commands.hs              FlexibleResize.hs
Roledex.hs               ThreeColumns.hs
CopyWindow.hs            FloatKeys.hs
RotSlaves.hs             TwoPane.hs
CycleWS.hs               FocusNth.hs
RotView.hs               ViewPrev.hs
DeManage.hs              HintedTile.hs
RunInXTerm.hs            Warp.hs
Decoration.hs            LICENSE
SetWMName.hs             WorkspaceDir.hs
DirectoryPrompt.hs       LayoutHelpers.hs
```

```
ShellPrompt.hs           XMonadPrompt.hs
Dmenu.hs                 LayoutHints.hs
SimpleDate.hs            XPrompt.hs
DragPane.hs              LayoutScreens.hs
SimpleStacking.hs        _darcs
DragPane.hs~             MagicFocus.hs
SinkAll.hs               scripts
DwmPromote.hs            Magnifier.hs
Spiral.hs                tests
DynamicLog.hs            MetaModule.hs
Square.hs
```

Harder to maintain discipline in the contrib library, but purity, haddock, public review are helping.

Idea: publish HPC coverage for extension modules in public: shame authors into testing

Helped avoid "window manager crank" syndrome: weird ideas are sketched out without breaking the core

## THE RELIABILITY TOOLKIT

Making it work

## THE RELIABILITY TOOLKIT

What's in the tool box?

- ➜ Cabal
- ➜ -Wall
- ➜ QuickCheck
- ➜ HPC
- ➜ Type system
- ➜ Catch

## CABAL!

If users can’t build the project, it is broken.

Cabal + Hackage have worked wonderfully here.

```
name:               xmonad
version:            0.3
homepage:           http://xmonad.org
synopsis:           A lightweight X11 window manager.
category:           System
license:            BSD3
license-file:       LICENSE
author:             Spencer Janssen
maintainer:         sjanssen@cse.unl.edu
build-depends:      base>=2.0, X11>=1.2.1, X11-extras>=0.3, mtl>=1.0, unix>=1.0

executable:         xmonad
main-is:            Main.hs
other-modules:      Config Operations StackSet XMonad
ghc-options:        -funbox-strict-fields -O2 -fasm -Wall -optl-Wl,-s
ghc-prof-options:   -prof -auto-all
extensions:         GeneralizedNewtypeDeriving
-- Also requires deriving Typeable
```

Very few problems reported with the build system!

## The enforcer of style, and guider of newbies..

```
ghc -Wall
```

Even better in 6.8

## QuickCheck

QuickCheck your window manager:

- ➜ Large suite of QC properties
- ➜ Must all pass before any commit
- ➜ Confirm they're effective with HPC
- ➜ Defines the semantics of the pure StackSet model library
- ➜ Helped guide the design: what is easy to QC, makes an intuitive UI

Only window manager with a testsuite?

## What can we test?

Start by generating random window manager models, then test some things:

- ➜ Invariants that are too hard to state in the type system
  - ➜ There are no duplicate windows in the StackSet
- ➜ Properties fully specifying the behaviour of each top level function
  - ➜ Moving focus doesn't affect window order
- ➜ Hunt for idempotent or reversible window manager behaviours
  - ➜ delete . insert is the identity on window managers
- ➜ Hunt for local invariants
  - ➜ Moving focus on one screen doesn't touch any other screens

## Checking QuickCheck with HPC

Use HPC to confirm your properties are complete and deep: checks your QuickChecks

- ➜ Won’t let you get away without full coverage
- ➜ Keeps you honest: exposes your ‘hand waving’ properties

Then enforce correctness by running QuickCheck and HPC on every commit.

- ➜ No patches going in if properties break.
- ➜ No features going in without properties.

Enforce 100% coverage!

## Enforce properties with types

Bug reports often resolved by relying on the type system

```
-- Code that might get an empty workspace
type StackOrNot a = Maybe (Stack a)

-- New numeric types (with deriving) to prevent indexing errors
newtype ScreenId    = S Int deriving (Eq,Ord,Num,Integral,Real)

-- Non-empty lists for layouts: force user’s hand
data Empty
data NonEmpty
data List x y where Nil :: List a Empty
                    Cons:: a -> List a b ->  List a NonEmpty
```

## CATCH: CHECKING FOR PATTERN-MATCH SAFETY

We regularly haul out Neil Mitchell's 'Catch' program to check the pure core of xmonad is free from pattern match failures and other partial definitions.

Strongly discourages the use of 'error'! Good practice.

When we first ran Catch we found:

- ➜ view calls error, this clearly makes it possible to crash.
- ➜ raiseFocus calls view
- ➜ promote calls head, in a potentially unsafe manner
- ➜ swap calls tail, again potentially unsafe.
- ➜ index calls fromJust, which means that it may crash.
- ➜ Ord instances are relied upon in a potentially unsafe way

a -Wpartial analysis for GHC would be really useful...

## CATCH: CHECKING FOR PATTERN-MATCH SAFETY

We regularly haul out Neil Mitchell's 'Catch' program to check the pure core of xmonad is free from pattern match failures and other partial definitions.

Strongly discourages the use of 'error'! Good practice.

When we first ran Catch we found:

- ➜ view calls error, this clearly makes it possible to crash.
- ➜ raiseFocus calls view
- ➜ promote calls head, in a potentially unsafe manner
- ➜ swap calls tail, again potentially unsafe.
- ➜ index calls fromJust, which means that it may crash.
- ➜ Ord instances are relied upon in a potentially unsafe way

a -Wpartial analysis for GHC would be really useful...

## Kicking butt with Haskell

We have all the weapons we need!

- ➜ Model effectful systems in purely functional data structures
- ➜ Use QuickCheck as your design assistant
- ➜ Use HPC to keep QuickCheck honest
- ➜ Enforce code quality with serious testing on every commit
- ➜ Don't be tempted by partial functions
- ➜ *Don't be tempted by side effects*
- ➜ Be responsive to bug reports
- ➜ Look at your competition's bugs, audit and prevent them

Gives you the space to move your application faster and further

## LIVE THE HASKELL VISION

code is more fun when it works!

xmonad.org