Software design and packaging for extensibility, provenance, and sharing

Jed Brown jedbrown@mcs.anl.gov

CIG Webinar, 2014-11-13

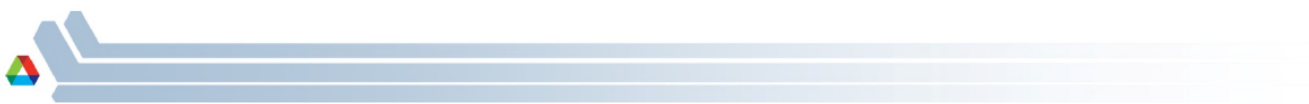
This talk: http://59A2.org/files/20141113-Software.pdf

Firetran!

- Renders HTML 10% faster than Firefox or Chromium.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page.

Firetran!

- Renders HTML 10% faster than Firefox or Chromium.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page.



Firetran!

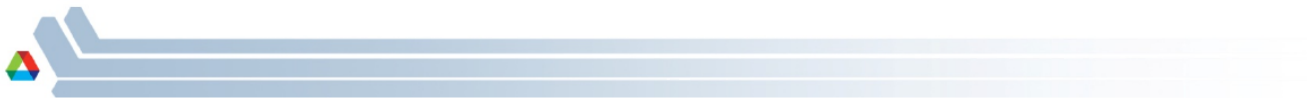
- Renders HTML 10% faster than Firefox or Chromium.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page.

Firetran!

- Renders HTML 10% faster than Firefox or Chromium.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page.

Firetran!

- Renders HTML 10% faster than Firefox or Chromium.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page.



Firetran!

- Renders HTML 10% faster than Firefox or Chromium.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page.

Firetran!

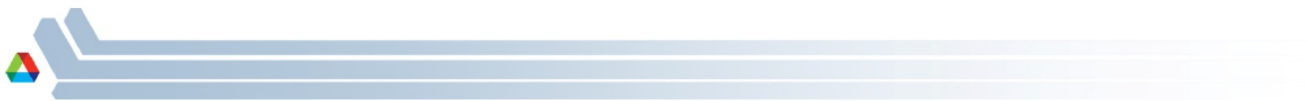
- Renders HTML 10% faster than Firefox or Chromium.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page.

Firetran!

- Renders HTML 10% faster than Firefox or Chromium.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page.

Firetran!

- Renders HTML 10% faster than Firefox or Chromium.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page.



Firetran struggles with market share

- Status quo in many scientific software packages
- Why do we tolerate it?
- Is scientific software somehow different?

Trends in Computational Science

- multiphysics, multiscale
- data assimilation, inversion, UQ
- risk-aware design and decision
- deeper software stacks
- many forms of extensibility
- artificial bottlenecks

Compile-time configuration

- configuration in build system
- over-emphasis on “efficiency”
- templates are compile-time
  - combinatorial number of variants
- compromises on-line analysis capability
- create artificial IO bottlenecks
- offloads complexity to scripts and “workflow” tools
- limits automation and testing of calibration
- maintaining consistency complicates provenance

Model coupling

- Hero codes
  - visionary scientist in single domain
  - each package is king of its own environment
- holes in knowledge exist at gaps between existing models
- models operate at different scales with different uncertainties
- coupling is hard enough with well-behaved components
- think like a library developer
  - minimize assumptions about environment
  - no globals, act locally, be explicit
  - successes: compilers, web browsers, databases

Provenance and Usability

- How to capture all configuration knobs so experiment can be reproduced? Compare
  - single run-time configuration file
  - compile-time configuration, multiple build systems, files passed between stages
- transitive dependencies must also be good libraries
- plugins better than source modification

“Big” Data

- Workflows with multiple executables pass data through file system
- About 1 hour to read/write contents of volatile memory
- Global storage as *alogrithmic* mechanism is dead
  - Better to run in-core on a larger machine
  - Out-of-core on full machine blows annual compute budget in one shot
- Circumvent IO bottleneck by passing data in-memory to next stage

Nested dependencies

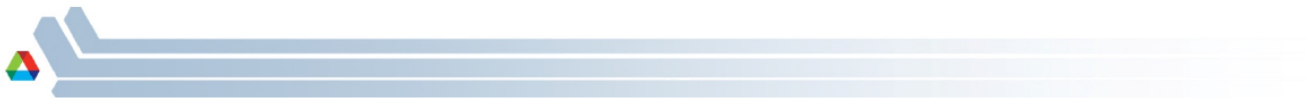
- Encapsulation is important to control complexity
- Reconfiguring indirect dependencies breaks encapsulation
- Single library may be used by multiple components in executable
  - diamond dependency graph
  - conflict unless same version/configuration can be used for both

Packaging and distribution

- Developers underestimate challenge of installing software
- User experience damaged even when user's fault (broken environment)
- Package managers (Debian APT, RedHat RPM, MacPorts, Homebrew, etc.)
- Binary interface stability critical to packagers

User modifications versus plugins

- Fragmentation is expensive and should be avoided
- Maintaining local modifications causes divergence
- Better to contain changes to a plugin
- `dlopen()` and register implementations in the shared library
- Invert dependencies and avoid loops
  - `libB` depends on `libA`
  - want optional implementation of `libA` that uses `libB`
  - `libA-plugin` depends on both `libA` and `libB`
- Static libraries are anti-productive (tell your computing center)
  - Can sort-of do plugins by changing link line

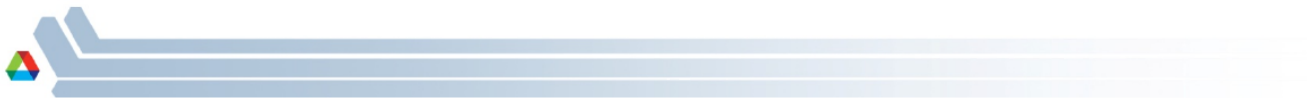


Controlling transitive complexity

- Implementation complexity must not leak into public interface
- Choose good defaults and provide a way to configure inner parts
- Inversion of Control (“dependency injection”, “service locator”)
- Can be multiple instances of components; identify using “prefix” rule
- Some use embedded Turing-complete configuration/scripting language

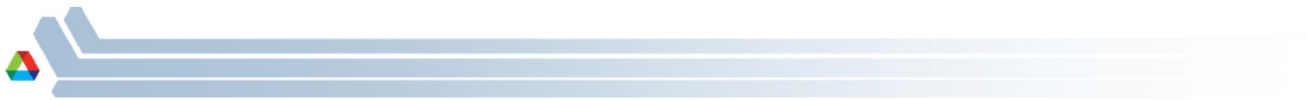
Object-oriented design

- Should all errors be compile-time errors?
- Sounds good in theory, but brittle
  - Should matrices have computable entries?
  - Should the diagonal be extractable?
  - Can the transpose be applied?
  - Do “Neumann” subproblems exist?
  - Different preconditioners require different properties from `Matrix`



Controlling the Binary interface

- Recompiling code is wasted productivity
- Implementation concerns (private variables, new virtual methods) should not require recompiling user code
- PETSc uses opaque pointers and the “delegator” (aka. “pointer to implementation”) pattern.
- Static function overhead insignificant, incremental cost less than 2 cycles
- Better for debugging
- Easier to expose libraries to dynamic programming languages



Just-in-Time Compilation (JIT)

- Fine-grained composition benefits from inlining
- Dynamic dispatch a much better library interface
- Templating not extensible via plugins, bloated, slow to compile
- JIT is promising for dynamic kernel fusion, plugin-style packaging

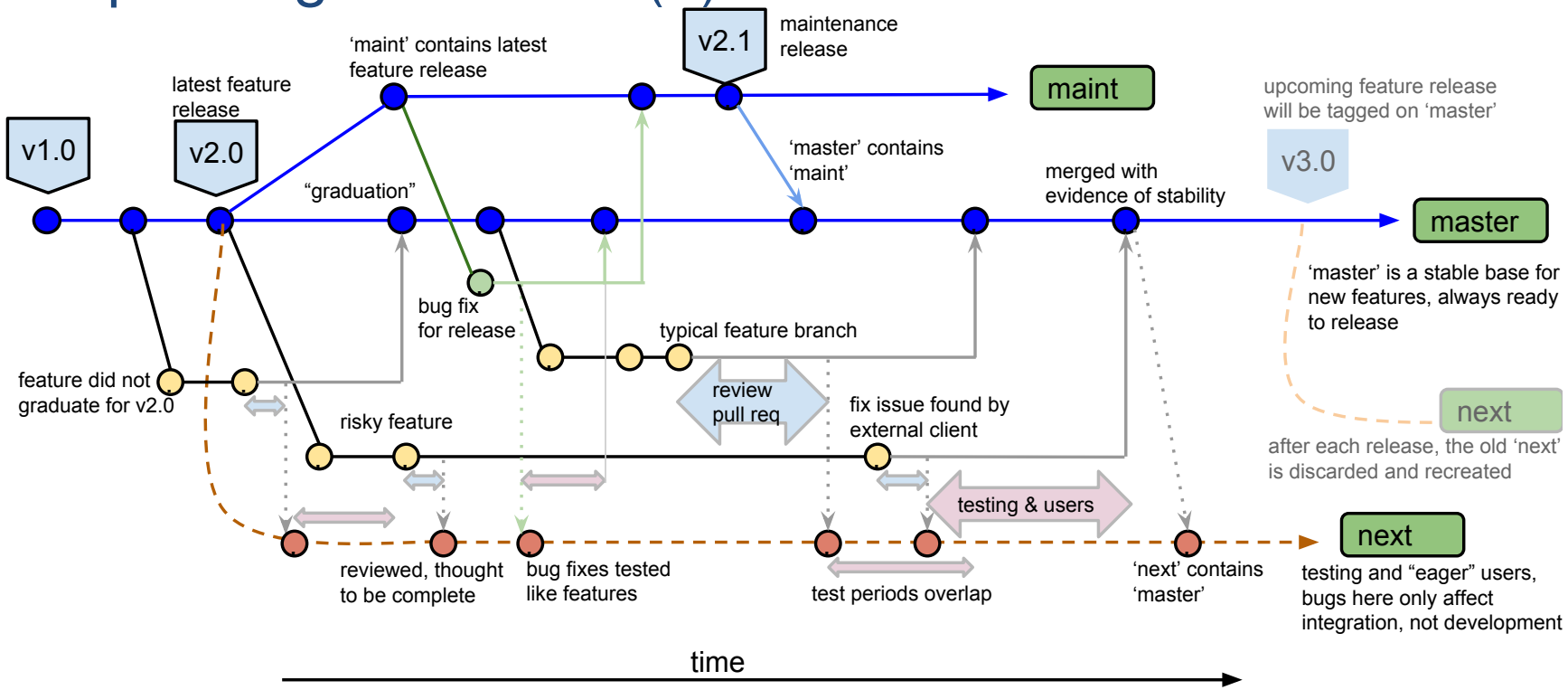
Upstreaming and community building

- Maintainers should provide good alternatives to forking
- Welcoming environment for contributions
- Privacy, “scooping”, openness
  - My opinion: social problem, deal with using social means
- Major tech companies have grossly underestimated cost of forking
- In science, we cannot pay off technical debt incurred by forking
- Provide extension points to reduce cost of new development

Workflow ideals

- 'master' is always stable and ready to release
- features are complete and tested before appearing in 'master'
- commits are minimal logically coherent, reviewable, and testable units
- related commits go together so as to be reviewable and debuggable by specialist
- new development is not disrupted by others' features and bugs
- rapid collaboration between developers possible
- `git log --first-parent maint..master` reads like a changelog
- bugs can be fixed once and anyone that needs the fix can obtain it without side-effects

Simplified gitworkflows(7)



first-parent history of branch

merge history (not first-parent)

merges to be discarded when 'next' is rewound at next release

merge in first-parent history of 'master' or 'maint' (approximate "changelog")

merge to branch 'next' (discarded after next major release)

commit in feature branch (feature branches usually start from 'master')

commit in bug-fix branch (bug-fix branches usually start from 'maint' or earlier)

Best practices

- Every branch has a purpose
- Distinguish integration branches from topic branches
- Do all development in topic branches
  - `git checkout -b my/feature-branch master`
- Namespace your branches if working on a shared repository
- Merge integration branches “forward”
  - `maint-1` → `maint` → `master` → `next`
  - `git checkout -b my/bugfix-branch maint-1`
- Write clear commit messages for reviewers and people trying to debug your code
- Avoid excessive merging from upstream
  - Always write a clear commit message explaining what is being merged and why
- Always merge topic branches as non-fast-forward (`merge --no-ff`)
- Gracefully retry if you lose a race to shared integration branch
  - This maximizes utility of `--first-parent` history

Outlook

- Think like a library developer
- Avoid assumptions about environment
- Make everything a run-time decision
- Control complexity
- Encourage contributions
- Plan for creative new directions you didn’t think of