

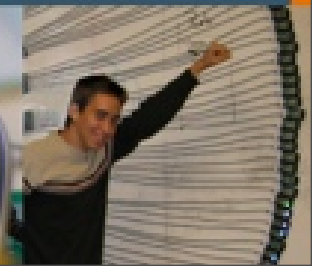
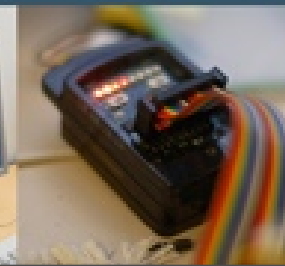


Introduction to the Programming Language Research Group

Eric Allen
Sun Labs



2008
Sun Labs
Open House



Copyright 2008 Sun Microsystems, Inc. ("Sun").

All rights are reserved by Sun except as expressly stated as follows. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted, provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers, or to redistribute to lists, requires prior specific written permission of Sun.

Project Fortress

- We are developing a programming language that meets the needs of the 21st century
 - > Increased emphasis on parallelism
 - > Able to grow with changing needs
 - > Well suited to programming-in-the-large
 - > Sophisticated syntax, closer to problem domains

Project Fortress

- Fortran has been around for over 50 years
- We are attempting to build an artifact that lasts for at least that long

Project Fortress

- This is an ambitious goal
- Achieving it requires passion, engagement, and frank discussion & evaluation
- Participate!
- Don't hold back your impressions!

Our Three-Pronged Approach

- We are developing a specification, implementation, and libraries in parallel
- Each task helps us to get the others right
- You will be responsible for an implementation task and a library task
- You are also invited to participate in improving the language design

Work as Part of a Team

- Don't be afraid to admit what you don't know
- Don't be afraid to admit what you can't do
- Don't hide how long you think it should take
- Don't hide what you haven't done yet
- Don't place blame; solve problems
- If you see a problem, speak up!

Work as Part of a Team

- Help your team members
- Don't disparage others for not knowing what you know
- Be ready and willing to teach what you know

Communication

- The life of a programming project depends on the continuity of the development team
- We need to transfer knowledge of various parts of the code

Communication

- Pair programming
- Skype
- Team meetings
- Boot Camp

Pair Programming

- Driver:
 - > Actually write the code
- Navigator:
 - > Review code as its written
 - > Keep the big picture in mind
 - > Retrieve information (such as library documentation) to assist the driver

Pair Programming

- Switch jobs between driver and navigator
- Switch partners from time to time
- By default, program in a pair
- This goes for Java programming and for Fortress programming!

Pair Programming

- Transfers knowledge of the code base
- Transfers general programming knowledge
- Keeps work focused on the task at hand
- Improves code quality
- More than pays for itself

Skype

- Skype allows us to communicate effectively as a distributed team
 - > Burlington, Austin, Houston, Denmark
- Be available on Skype when working at a computer and not pair programming
 - > It's okay to hide on occasion
- Use Skype to get quick answers to your questions

Team Meetings

- We have a team meeting every Monday morning at 9:30 AM CT/10:30 AM ET
- Be prepared to report on what you accomplished over the past week and to discuss problems

Technical Sessions

- We debate modifications to the language
- Sukyoung maintains the agenda
- When you have an opinion, speak up!
- Feel free to propose language changes on the language@projectfortress.sun.com mailing list for discussion at these meetings
- External proposals on the list must be championed by a meeting participant to be discussed

Boot Camp

- The purpose of Fortress Boot Camp is to allow you to “hit the ground running” on week two
- Learn what you need to learn to start your summer project

Implementation Tasks

- Nels, Justin: Complete type checking and inference
- Jon, Ryan: Complete syntax abstraction
- Michael, Angelina: Implement efficient environments and an encoding of Fortress types into Java types

Library Tasks

- This week, choose a library task you are interested in
- Consider tying this task to your implementation work if possible

Consult the Spec

- When you are implementing a language feature, find the definition of its behavior in the specification
- This serves two purposes:
 - > It helps us get the implementation right
 - > It helps us improve the spec

Test-Driven Development

- Start with a few tests
- Write the simplest code you can to pass those tests
- If your code doesn't do everything you want it to, *write more tests and repeat*

Writing Code and Tests in Tandem

- Tests are a great source of documentation
 - > They are precise
 - > They are executable and always in synch
- They help us keep the design simple
- They help us make incremental progress
- They protect your code

Never commit code without first:

- Doing an “svn up”
- Doing an “svn st” and ensuring that all new files and directories have been added
- Running “./ant clean test” and ensuring that all tests are passing

Commit Code Often

- Aim to commit at least once a day
 - > Akin to balancing a check book
- Break implementation/testing tasks up into small enough chunks that this is feasible
- This constraint might affect your design and the order of your implementation tasks

***Help to improve the language,
and to make it your own.***