Elmer

Software Development Practices
APIs for Solver and UDF

ElmerTeam
CSC – IT Center for Science

CSC, November.2015

Elmer programming languages

- Fortran90 (and newer)
  - ElmerSolver (~240,000 lines of which ~50% in DLLs)
- C++
  - ElmerGUI (~18,000 lines)
  - ElmerSolver (~10,000 lines)
- C
  - ElmerPost (~45,000 lines)
  - ElmerGrid (~30,000 lines)
  - MATC (~11,000 lines)

Tools for Elmer development

- Programming languages
  - Fortran90 (and newer), C, C++
- Compilation
  - Compiler (e.g. gnu), configure, automake, make, (cmake)
- Editing
  - emacs, vi, notepad++,…
- Code hosting (git)
  - Current: https://github.com/ElmerCSC
  - Obsolite: www.sf.net/projects/elmerfem
- Consistency tests
- Code documentation
  - Doxygen
- Theory documentation
  - Latex
- Community server
  - www.elmerfem.org (forum, wiki, etc.)

Elmer libraries

- ElmerSolver
  - Required: Matc, HutIter, Lapack, Blas, Umfpack (GPL)
  - Optional: Arpack, Mumps, Hypre, Pardiso, Trilinos, SuperLU, Cholmod, NetCDF, HDF5, …
- ElmerGUI
  - Required: Qt, ElmerGrid, Netgen
  - Optional: Tetgen, OpenCASCADE, VTK, QVT

Elmer licenses

- ElmerSolver library is published under LGPL
  - Enables linking with all license types
  - It is possible to make a new solver even under proprietary license
  - Note: some optional libraries may constrain this freedom due to use of GPL licences
- Rest of Elmer is published under GPL
  - Derived work must also be under same license (“copyleft”)

Elmer version control at GitHub

- In 2015 the official version control of Elmer was transferred from svn at sf.net to git hosted at GitHub
- Git offers more flexibility over svn
  - Distributed version control system
  - Easier to maintain several development branches
  - More options and hence also steeper learning curve
  - Developed by Linus Torvalds to host Linux kernel development
- GitHub is a portal providing Git and some additional services
  - Management of user rights
  - Controlling pull requests

Git- Version control system

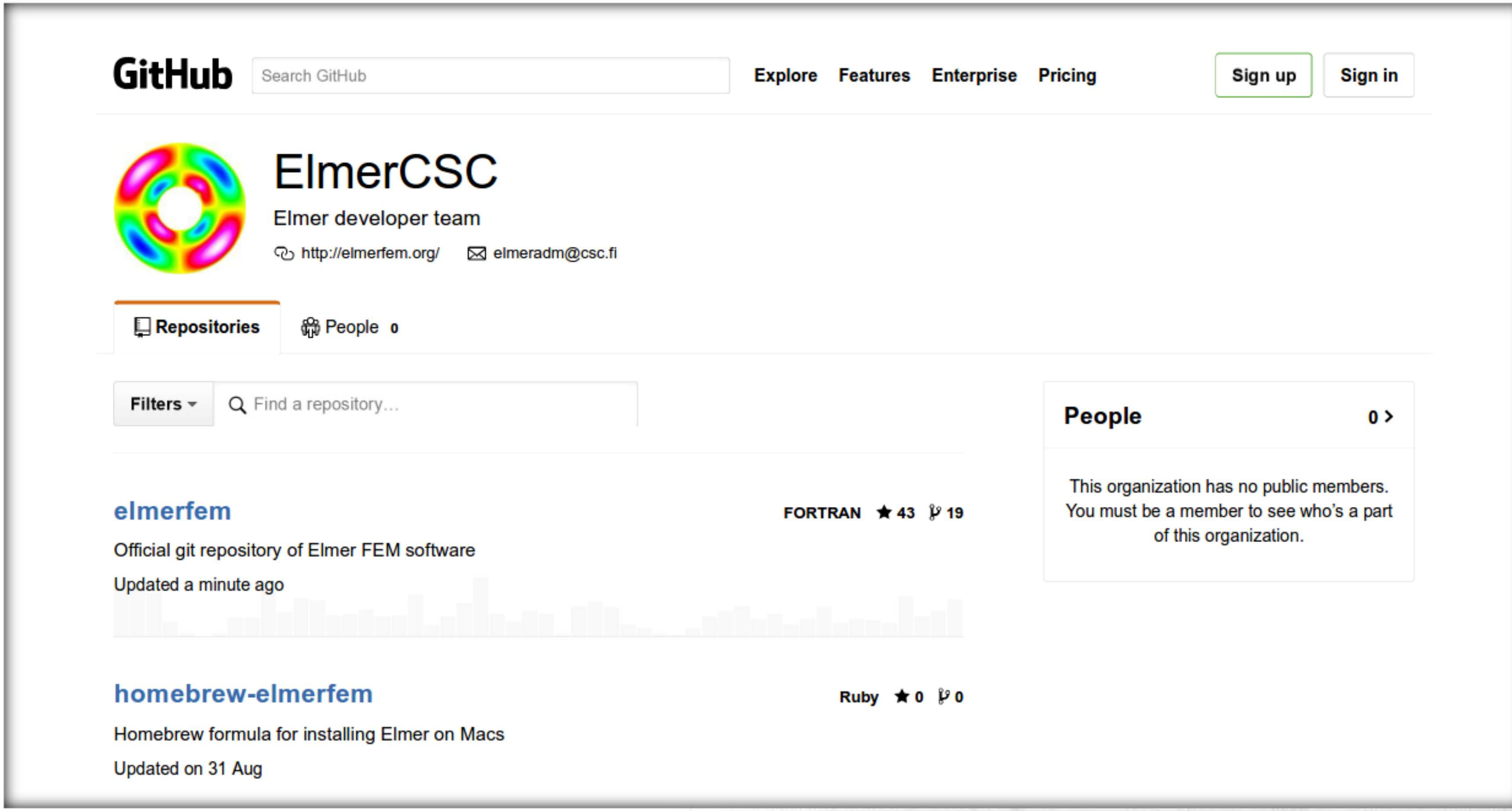
- Elmer uses git version control system for the code repository and development
  - Hosted at github
  - Development version in ”trunk” is considered stable
  - To obtain the whole source code

```
git clone https://github.com/ElmerCSC/elmerfem.git
```

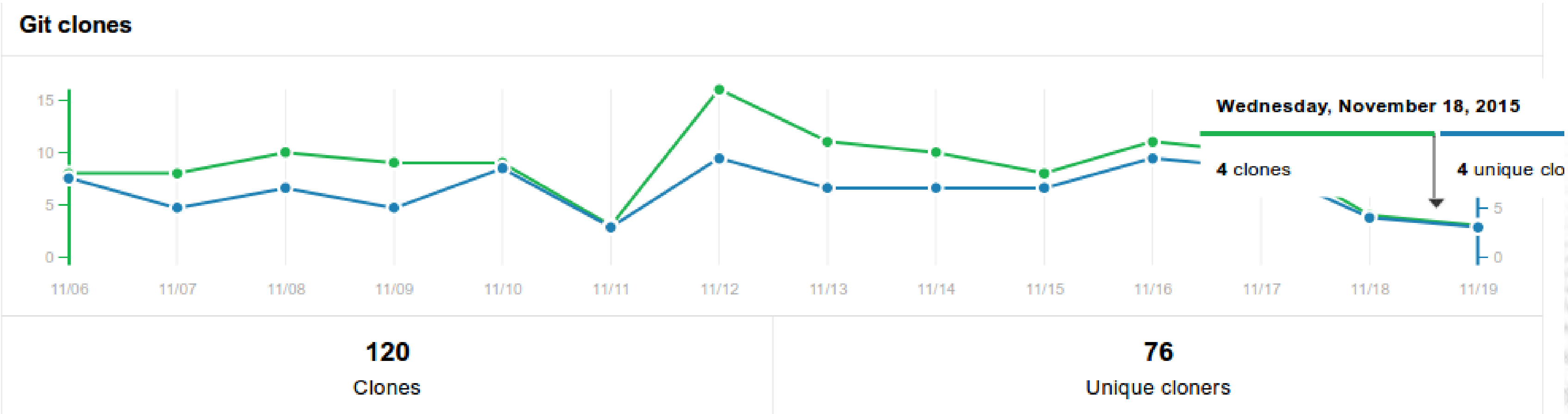
  - Git client available in command line in *nix systems
  - In Windows systems a nice graphical client is “Tortoise”

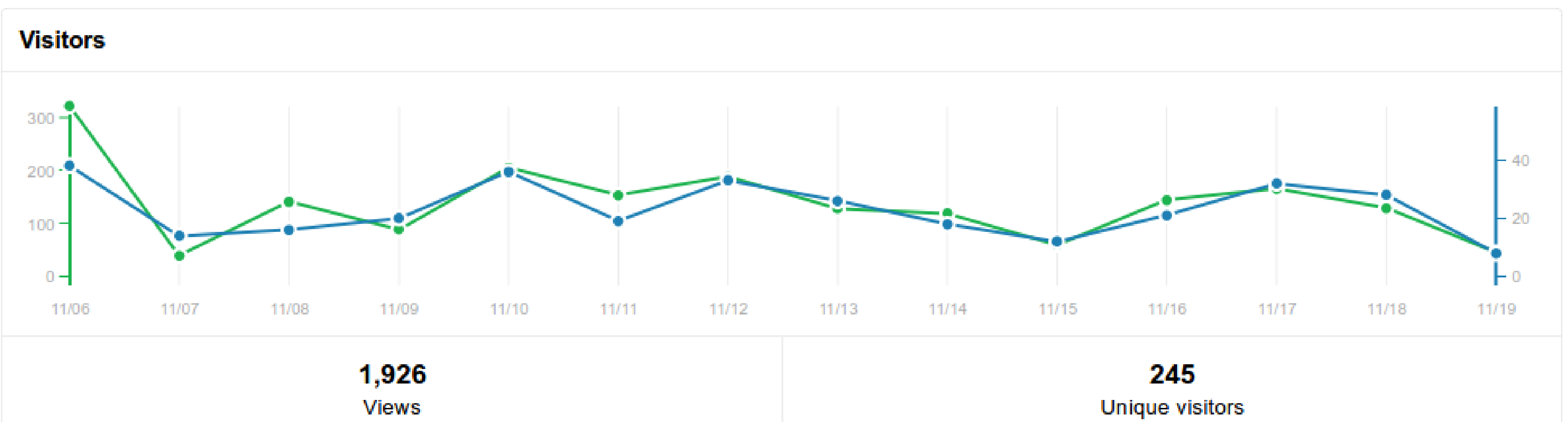
Elmer at Github

https://github.com/ElmerCSC



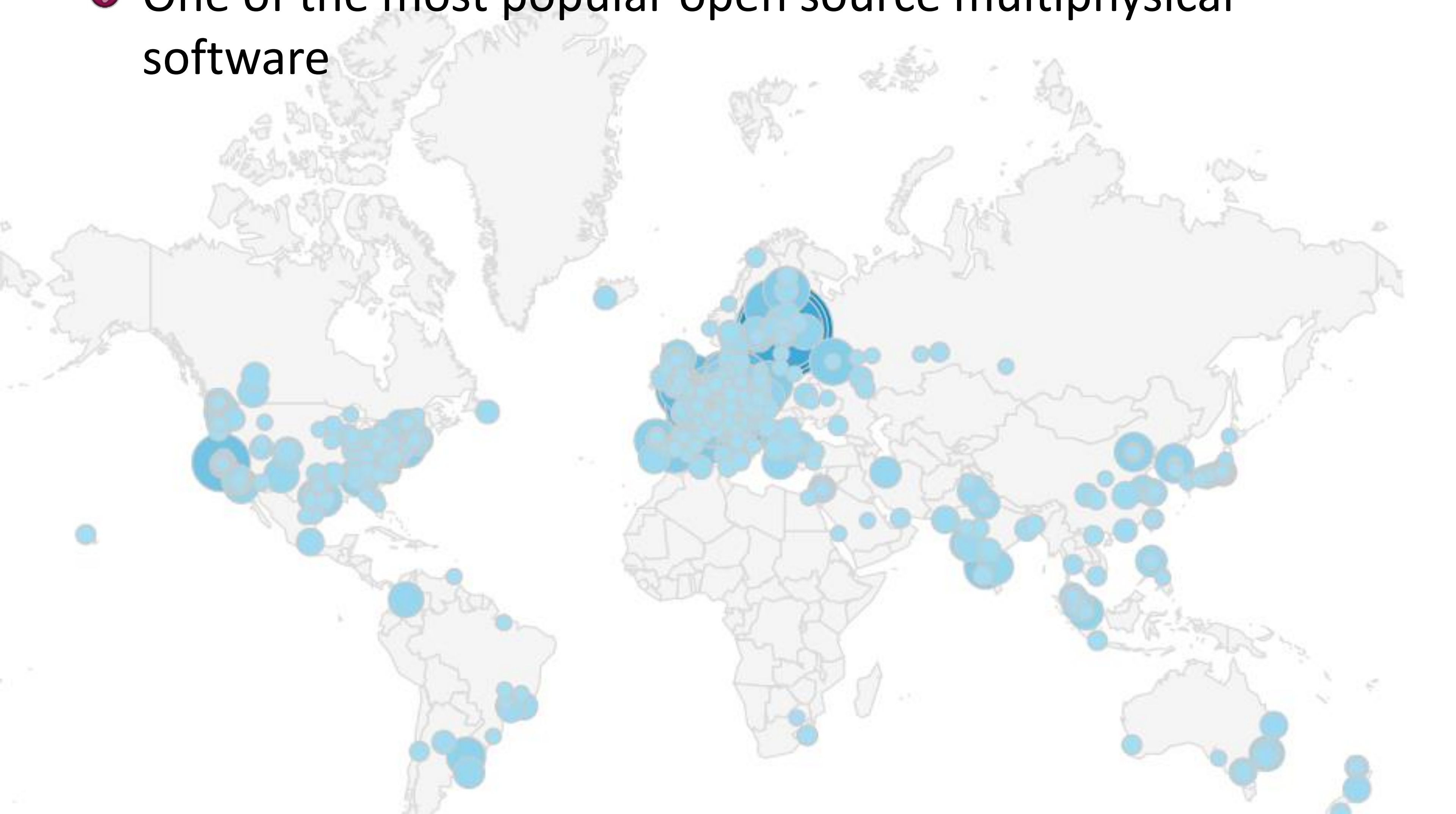
Activity on Github





Elmer is published under (L)GPL

- Used worldwide by thousands of researchers (?)
- One of the most popular open source multiphysical software

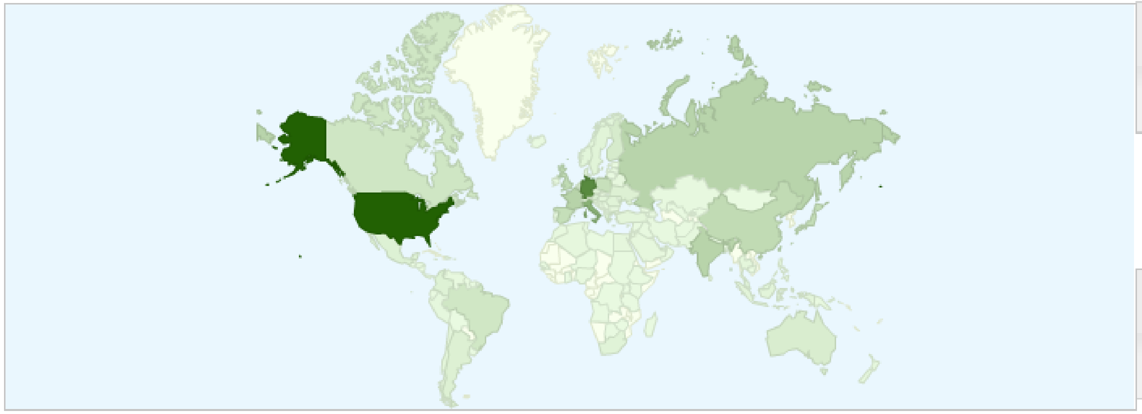


	Country / Territory	Visits	Pages / Visit	Avg. Visit Duration	% New Visits	Bounce Rate
1.	United States	**9,611**	3.52	00:03:31	55.59%	60.55%
2.	Germany	**8,938**	5.08	00:05:52	31.76%	51.92%
3.	France	**4,404**	3.15	00:03:06	30.68%	68.66%
4.	United Kingdom	**4,028**	4.19	00:04:27	39.03%	53.87%
5.	Finland	**3,841**	7.06	00:09:14	20.57%	22.49%
6.	Italy	**3,602**	4.44	00:03:38	55.19%	47.47%
7.	Spain	**1,957**	6.16	00:05:52	36.54%	46.65%
8.	Japan	**1,484**	3.14	00:03:20	36.59%	64.08%
9.	India	**1,341**	2.86	00:02:39	73.38%	69.57%
10.	Canada	**1,335**	3.12	00:02:53	58.13%	63.00%
11.	Netherlands	**1,333**	3.73	00:03:44	37.21%	58.21%
12.	Austria	**1,120**	4.88	00:04:43	29.29%	53.04%
13.	Switzerland	**995**	3.63	00:03:23	40.80%	59.30%
14.	China	**975**	4.14	00:04:04	48.92%	53.54%
15.	Poland	**907**	4.06	00:03:59	41.23%	55.35%
16.	Brazil	**876**	2.98	00:03:16	42.01%	64.50%
17.	Russia	**831**	2.97	00:02:53	51.50%	65.82%

~20k Windows downloads at sf.net in a year

Home / WindowsBinaries (Change File)

Date Range: 2012-04-01 to 2013-03-31



DOWNLOADS
19 185
In the selected date range

TOP COUNTRY
United States
16% of downloaders

TOP OS
Windows
93% of downloaders

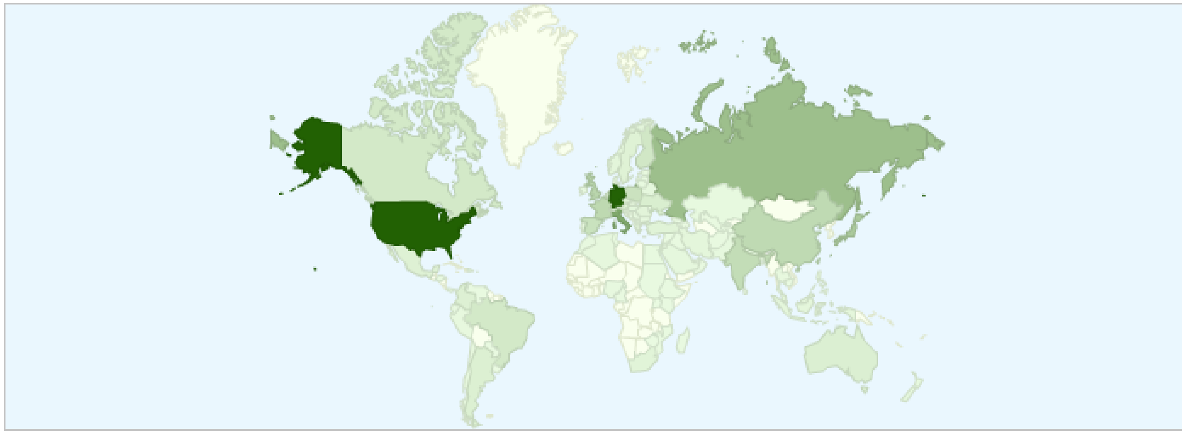
OS downloads as: Percent

	Country	Android	BSD	Linux	Macintosh	Unknown	Windows	Total
1.	United States	0%	0%	3%	3%	1%	80%	3,182
2.	Germany	0%	0%	4%	1%	0%	80%	2,313
3.	Italy	0%	0%	3%	1%	0%	80%	1,537
4.	France	0%	0%	4%	1%	1%	79%	798
5.	India	0%	0%	6%	1%	4%	78%	782
6.	Russia	0%	0%	4%	0%	0%	77%	772
7.	United Kingdom	0%	0%	3%	2%	0%	81%	642
8.	China	0%	0%	3%	1%	1%	78%	637
9.	Japan	0%	0%	2%	2%	0%	77%	599
10.	Spain	0%	0%	6%	0%	20%	63%	561
11.	Poland	0%	0%	2%	0%	0%	87%	532
12.	Canada	1%	0%	2%	2%	0%	85%	410
13.	Brazil	0%	0%	4%	1%	0%	88%	391
14.	Finland	0%	0%	2%	1%	0%	78%	300

16k Windows downloads at sf.net in a year

Home / WindowsBinaries (Change File)

Date Range: 2011-06-01 to 2012-06-01



DOWNLOADS
16 214
In the selected date range

TOP COUNTRY
United States
15% of downloaders

TOP OS
Windows
94% of downloaders

	Country	Downloads
1.	United States	2,553
2.	Germany	2,529
3.	Italy	1,342
4.	Russia	975
5.	Japan	789
6.	United Kingdom	609
7.	France	548
8.	China	529
9.	India	483
10.	Spain	400
11.	Poland	385
12.	Finland	305

Installers

- Fresh Windows installers
  - Currently only 64 bit version
  - Also a parallel version with msmpi
  - http://www.nic.funet.fi/pub/sci/physics/elmer/bin/windows/
- Elmer for Debian & Ubuntu etc. at launchpad
  - Nightly builds from Git repository
  - To install

```
$ sudo apt-add-repository ppa:juhmat/elmer-test
$ sudo apt-get update
$ sudo apt-get install elmerfem-csc
```

Cmake build system

- During 2014-2015 Elmer was migrated from gnu autotools to **cmake**
- cmake offers several advantages
  - Enables cross compilation for different platforms (e.g. Intel MICs)
  - More standardizes installation scripts
  - Straight-forward package creation for many systems (using cpack)
  - Great testing utility with ctest
- Transition to cmake required significant code changes
  - ISO C-bindings & many changes in APIs
  - Backward compatibility in compilation lost

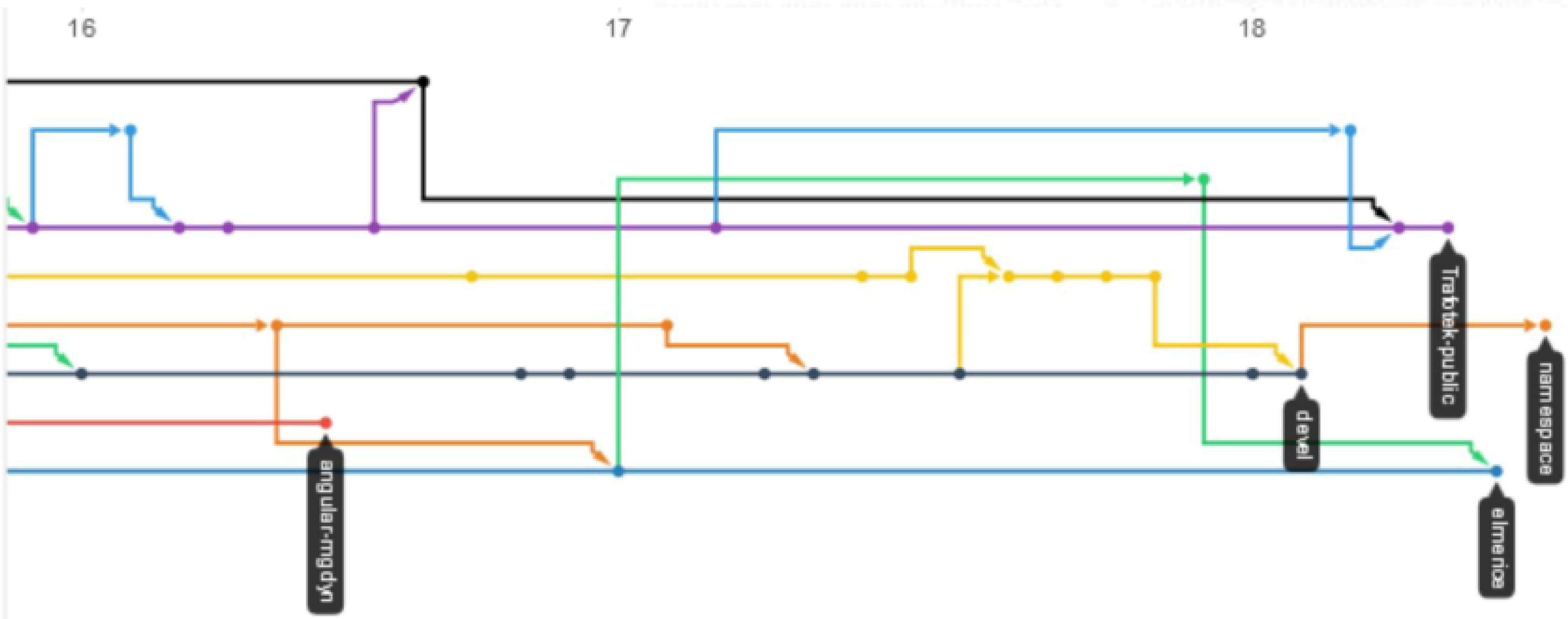
Obtaining the source code

- To clone the code (this is anonymously):
```
git clone \
https://github.com/ElmerCSC/elmerfem.git
```
  - We work with branches. To change into another branch:
```
cd elmerfem
git checkout branchname
```
- We use the following branches (confined to most important):
  - ***release***: contains stable release (~half-yearly update)
  - ***devel***: our main branch from which you should get latest updates
  - ***elmerice***: the main developer branch for Elmer/Ice

    NB: you might have to do a
```
git checkout --track origin/elmerice
```

Obtaining the source code

- On branches:



Code organization

fem	Source of ElmerSolver
matc	MATC language
fhutiter	Fortran version of linear algebra solvers
ElmerGUI	Graphical User Interface t Elmer based on QT4
elmerice	Elmer/Ice solver and function source code
post	Legacy visualization tool
elmergrid	Grid manipulation for Elm
mathlibs	Contains Lapack/BLAS fro netlib (avoid using them)
elmerparam	Additional package for optimization

- ElmerGUI
- ElmerGUIlogger
- ElmerGUItester
- buildtools
- cmake
- cpack
- eio
- elmergrid
- elmerice
- elmerparam
- fem
- fhutiter
- front
- hutiter
- license_texts
- matc
- mathlibs
- meshgen2d
- misc
- post
- umfpack

Compilation of the whole code with cmake

- To compile the whole code see example scripts under www.csc.fi/elmer and www.elmerfem.org

```
ELMERSRC="/path/to/sourcecode/elmerfem"
BUILDDIR="/path/too/existing/and/empty/builddir"
IDIR="/path/to/installation/dir/”
TOOLCHAIN="/path/to/optional/toolchainfile.smake"
cmake $ELMERSRC \
    -DCMAKE_BUILD_TYPE=DEBUG\
    -DCMAKE_INSTALL_PREFIX=$IDIR \
    -DWITH_MPI:BOOL=TRUE \
    -DWITH_Mumps:BOOL=TRUE \
    -DWITH_Hypre:BOOL=TRUE \
    -DWITH_ELMERGUI:BOOL=TRUE \
    -DWITH_OCC:BOOL=TRUE \
    -DWITH_PARAVIEW:BOOL=TRUE \
    -DWITH_PYTHONQT:BOOL=TRUE \
    -DWITH_QWT:BOOL=TRUE \
    -DWITH_VTK:BOOL=TRUE \
    -DWITH_PYTHONQT:BOOL=FALSE \
    -DWITH_MATC:BOOL=TRUE \
    -DWITH_ElmerIce:BOOL=TRUE
```

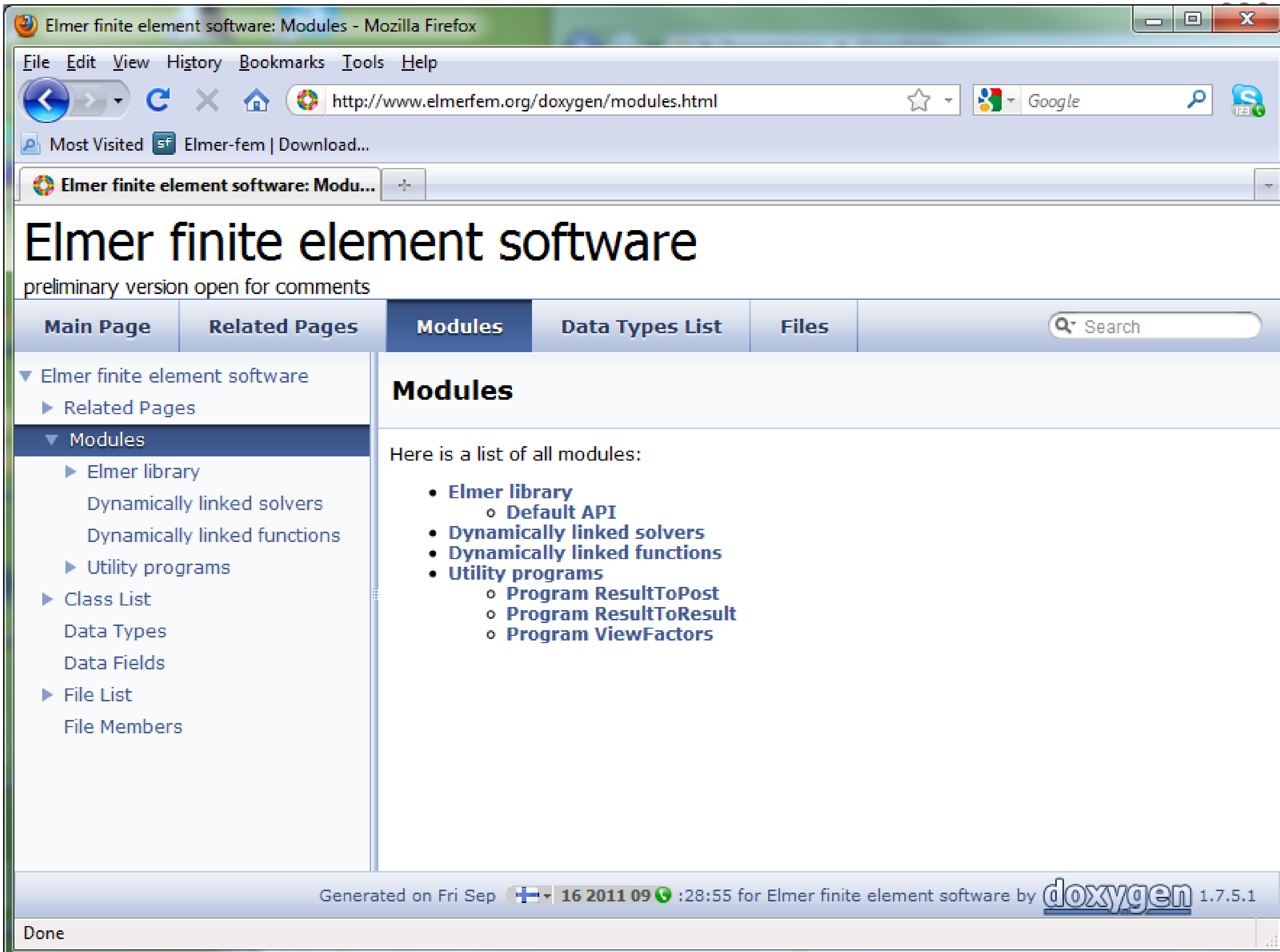
Consistency tests

- Simple shell script to run through the cases + piece of C-code to compare the norm of solutions
- There are >300 consistency tests (November 2015)
  - Located under fem/tests, run with ctest in build-directory
- Each time a significant commit is made the tests are run with the fresh version
  - Aim: trunk version is a stable version
  - New tests for each major new feature
- The consistency tests provide a good starting point for taking some Solver into use
  - cut-paste from sif file
- Note: the consistency tests have often poor time and space resolution for rapid execution

Consistency tests - example

```
Terminal
zwinger@elmeruser-VM64bit ~/Source/Elmer_devel/elmerfem $ cd ..
zwinger@elmeruser-VM64bit ~/Source/Elmer_devel $ cd builddir_elmerice/
zwinger@elmeruser-VM64bit ~/Source/Elmer_devel/builddir_elmerice $ ctest -LE elm
erice
Test project /home/zwinger/Source/Elmer_devel/builddir_elmerice
        Start  35: mgdyn_lamstack_widefreq_harmonic
  1/379 Test  #35: mgdyn_lamstack_widefreq_harmonic ......   Passed   12.58 sec
        Start  36: NaturalConvectionRestart
  2/379 Test  #36: NaturalConvectionRestart ..............   Passed   10.90 sec
        Start  37: ContactPatch3D
  3/379 Test  #37: ContactPatch3D ........................   Passed    4.28 sec
        Start  38: ContactPatch3D_np4
  4/379 Test  #38: ContactPatch3D_np4 ....................   Passed    6.13 sec
        Start  39: CurvedBndryPFEM
  5/379 Test  #39: CurvedBndryPFEM .......................   Passed    0.12 sec
        Start  40: heateq
  6/379 Test  #40: heateq ................................   Passed    0.42 sec
        Start  41: StrainCalculation02
  7/379 Test  #41: StrainCalculation02 ...................   Passed    8.62 sec
        Start  42: ContactPatch2Dtwo
  8/379 Test  #42: ContactPatch2Dtwo .....................   Passed    0.16 sec
        Start  43: freesurf
  9/379 Test  #43: freesurf ..............................   Passed    1.61 sec
        Start  44: rotflow
```

Doxygen – WWW documentation



Doxygen – Example in code

- Special comment indicators: !> and <!

```
!------------------------------------------------------------------------------
!> Subroutine for computing fluxes and gradients of scalar fields.
!> For example, one may compute the the heat flux as the negative gradient of temperature
!> field multiplied by the heat conductivity.
!> \ingroup Solvers
!------------------------------------------------------------------------------
SUBROUTINE FluxSolver( Model,Solver,dt,Transient )
!------------------------------------------------------------------------------
USE CoordinateSystems
USE DefUtils
IMPLICIT NONE
!------------------------------------------------------------------------------
TYPE(Solver_t) :: Solver  !< Linear & nonlinear equation solver options
TYPE(Model_t) :: Model    !< All model information (mesh, materials, BCs, etc...)
REAL(KIND=dp) :: dt       !< Timestep size for time dependent simulations
LOGICAL :: Transient      !< Steady state or transient simulation
!------------------------------------------------------------------------------
!    Local variables
!------------------------------------------------------------------------------
TYPE(ValueList_t),POINTER :: SolverParams
```

Doxygen – Example in WWW

```
subroutine FluxSolver ( TYPE(Model_t)    Model,
                        TYPE(Solver_t)   Solver,
                        REAL(KIND=dp)    dt,
                        LOGICAL          Transient
                      )
```

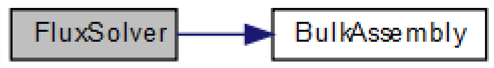
Subroutine for computing fluxes and gradients of scalar fields. For example, one may compute the the heat flux as the negative gradient of temperature field multiplied by the heat conductivity.

Parameters:

Solver	Linear & nonlinear equation solver options
Model	All model information (mesh, materials, BCs, etc...)
dt	Timestep size for time dependent simulations
Transient	Steady state or transient simulation

References **BulkAssembly()**.

Here is the call graph for this function:



Compilation of a DLL module

- Applies both to Solvers and User Defined Functions (UDF)
- Assumes that there is a working compile environment that provides ”**`elmerf90`**” script
  - Comes with the Windows installer, and Linux packages
  - Generated automatically when ElmerSolver is compiled

```
elmerf90 MySolver.f90 –o MySolver.so
```

User defined function API

```
!------------------------------------------------------
!> Standard API for UDF
!------------------------------------------------------
RECURSIVE FUNCTION MyProperty( Model, n, t ) RESULT(f)
!------------------------------------------------------
  USE DefUtils
  IMPLICIT NONE
!------------------------------------------------------
  TYPE(Model_t) :: Model      !< Handle to all data
  INTEGER :: n                !< Current node
  REAL(KIND=dp) :: t          !< Parameter(s)
  REAL(KIND=dp) :: f          !< Parameter value at node
!------------------------------------------------------
  Actual code …
```

Function API

```
MyProperty = Variable time
 "MyModule" "MyProperty"
```

- User defined function (UDF) typically returns a real valued property at a given point
- It can be located in any section that is used to fetch these values from a list
  - Boundary Condition, Initial Condition, Material,...

Solver API

```
!------------------------------------------------------
!> Standard API for Solver
!------------------------------------------------------
SUBROUTINE MySolver( Model,Solver,dt,Transient )
!------------------------------------------------------
  USE DefUtils
  IMPLICIT NONE
!------------------------------------------------------
  TYPE(Solver_t) :: Solver   !< Current solver
  TYPE(Model_t) :: Model     !< Handle to all data
  REAL(KIND=dp) :: dt        !< Timestep size
  LOGICAL :: Transient       !< Time-dependent or not
!------------------------------------------------------
  Actual code …
```

Solver API

```
Solver 1
   Equation = "MySolver"
   Procedure = "MyModule" "MySolver"
   …
End
```

- Solver is typically a FEM implementation of a physical equation (PDE)
- But it could also be an auxiliary solver that does something completely different
- Solver is usually called once for each coupled system iteration

Elmer – High level abstractions

- The quite good success of Elmer as a multi-physics code may be addressed to certain design choices
  - Solver is an abstract dynamically loaded object
  - Parameter value is an abstract property fetched from a list
- The abstractions mean that new solvers may be implemented without much need to touch the main library
  - Minimizes need of central planning
  - Several applications fields may live their life quite independently (electromagnetics and glaciology)
- MATC – a poor man's Matlab adds to flexibility as algebraic expressions may be evaluated on-the-fly

Solver as an abstract object

- Solver is a dynamically loaded object (.dll or .so)
  - May be developed and compiled separately
- Solver utilizes heavily common library utilities
  - Most common ones have interfaces in DefUtils
- Any solver has a handle to all of the data
- Typically a solver solves a weak form of a differential equation
- Currently ~50 different Solvers, roughly half presenting physical phenomena
  - No upper limit to the number of Solvers
- Solvers may be active in different domains, and even meshes
- The menu structure of each solver in ElmerGUI may be defined by an `.xml` file

Properties as abstract objects

- Properties are saved in a list structure by their name
- Namespace of properties is not fixed, they may be introduced in the command file
  - E.g. ”`MyProperty = Real 1.23`” adds a property ”MyProperty” to a list structure related to the solver block
- In the code parameters are fetched from the list
  - E.g. ”`val = GetReal( Material,'MyProperty',Found`)” retrieves the above value 1.23 from the list
- A ”`Real`” property may be any of the following
  - Constant value
  - Linear or cubic dependence via table of values
  - Expression given by MATC (MatLab/C-type command language)
  - User defined functions with arbitrary dependencies
  - Real vector or tensor
- As a result solvers may be weakly coupled without any *a priori* defined manner
- There is a price to pay for the generic approach but usually it is less than 10%
- `SOLVER.KEYWORDS` file may be used to give the types for the keywords in the command file

DefUtils

- DefUtils module includes wrappers to the basic tasks common to standard solvers
  - E.g. ”**`DefaultDirichlet()`**” sets Dirichlet boundary conditions to the given variable of the Solver
  - E.g. ”**`DefaultSolve()`**” solves linear systems with all available direct, iterative and multilevel solvers, both in serial and parallel
- Programming new Solvers and UDFs may usually be done without knowledge of other modules

DefUtils – some functions

Public Member Functions

TYPE(Solver_t) function, pointer	**GetSolver** ()
TYPE(Matrix_t) function, pointer	**GetMatrix** (USolver)
TYPE(Mesh_t) function, pointer	**GetMesh** (USolver)
TYPE(Element_t) function, pointer	**GetCurrentElement** (Element)
INTEGER function	**GetElementIndex** (Element)
INTEGER function	**GetNOFActive** (USolver)
REAL(KIND=dp) function	**GetTime** ()
INTEGER function	**GetTimeStep** ()
INTEGER function	**GetTimeStepInterval** ()
REAL(KIND=dp) function	**GetTimestepSize** ()
REAL(KIND=dp) function	**GetAngularFrequency** (ValueList, Found)
INTEGER function	**GetCoupledIter** ()
INTEGER function	**GetNonlinIter** ()
INTEGER function	**GetNOFBoundaryElements** (UMesh)
subroutine	**GetScalarLocalSolution** (x, name, UElement, USolver, tStep)
subroutine	**GetVectorLocalSolution** (x, name, UElement, USolver, tStep)
INTEGER function	**GetNofEigenModes** (name, USolver)
subroutine	**GetScalarLocalEigenmode** (x, name, UElement, USolver, NoEigen, ComplexPart)
subroutine	**GetVectorLocalEigenmode** (x, name, UElement, USolver, NoEigen, ComplexPart)
CHARACTER(LEN=MAX_NAME_LEN) function	**GetString** (List, Name, Found)
INTEGER function	**GetInteger** (List, Name, Found)
LOGICAL function	**GetLogical** (List, Name, Found)
recursive REAL(KIND=dp) function	**GetConstReal** (List, Name, Found, x, y, z)
recursive REAL(KIND=dp) function	**GetCReal** (List, Name, Found)
recursive REAL(KIND=dp) function, dimension(:), pointer	**GetReal** (List, Name, Found, UElement)

Example: Poisson equation

$$-\nabla^2\phi = \rho$$

- Implemented as an dynamically linked solver
  - Available under tests/1dtests
- Compilation by:
  `elmerf90 Poisson.f90 -o Poisson.so`
- Execution by:
  `ElmerSolver case.sif`

- The example is ready to go massively parallel and with all a plethora of elementtypes in 1D, 2D and 3D

Poisson equation: code Poisson.f90

```
!------------------------------------------------------------------------------
!> Solve the Poisson equation -\nabla\cdot\nabla \phi = \rho
!------------------------------------------------------------------------------
SUBROUTINE PoissonSolver( Model,Solver,dt,TransientSimulation )
!------------------------------------------------------------------------------
 USE DefUtils
 IMPLICIT NONE
 …

  !Initialize the system and do the assembly:
  !-----------------------------------------
  CALL DefaultInitialize()

  active = GetNOFActive()
  DO t=1,active
    Element => GetActiveElement(t)
    n = GetElementNOFNodes()

    LOAD = 0.0d0
    BodyForce => GetBodyForce()
    IF ( ASSOCIATED(BodyForce) ) &
      Load(1:n) = GetReal( BodyForce, 'Source', Found )

    ! Get element local matrix and rhs vector:
    !----------------------------------------
    CALL LocalMatrix(  STIFF, FORCE, LOAD, Element, n )

    ! Update global matrix and rhs vector from local contribs
    !---------------------------------------------------------------
    CALL DefaultUpdateEquations( STIFF, FORCE )
  END DO

  CALL DefaultFinishAssembly()
  CALL DefaultDirichletBCs()
  Norm = DefaultSolve()
```

```
Solver 1
  Equation = "Poisson"
  Variable = "Potential"
  Variable DOFs = 1
  Procedure = "Poisson" "PoissonSolver"
  Linear System Solver = "Direct”
  Linear System Direct Method = umfpack
  Steady State Convergence Tolerance = 1e-09
End
Body Force 1
  Source = Variable Potential
   Real Procedure "Source" "Source"
End

Boundary Condition 1
  Target Boundaries(2) = 1 2
  Potential = Real 0
End
```

Poisson equation: code Poisson.f90

```
CONTAINS

!------------------------------------------------------------------------------
 SUBROUTINE LocalMatrix(  STIFF, FORCE, LOAD, Element, n )
!------------------------------------------------------------------------------

…

CALL GetElementNodes( Nodes )
   STIFF = 0.0d0
   FORCE = 0.0d0

   ! Numerical integration:
   !----------------------
   IP = GaussPoints( Element )
   DO t=1,IP % n
     ! Basis function values & derivatives at the integration point:
     !--------------------------------------------------------------
     stat = ElementInfo( Element, Nodes, IP % U(t), IP % V(t), &
           IP % W(t),  detJ, Basis, dBasisdx )

     ! The source term at the integration point:
     !----------------------------------------
     LoadAtIP = SUM( Basis(1:n) * LOAD(1:n) )

     ! Finally, the elemental matrix & vector:
     !--------------------------------------
     STIFF(1:n,1:n) = STIFF(1:n,1:n) + IP % s(t) * DetJ * &
         MATMUL( dBasisdx, TRANSPOSE( dBasisdx ) )
     FORCE(1:n) = FORCE(1:n) + IP % s(t) * DetJ * LoadAtIP * Basis(1:n)
   END DO
!------------------------------------------------------------------------------
 END SUBROUTINE LocalMatrix
!------------------------------------------------------------------------------
END SUBROUTINE PoissonSolver
!------------------------------------------------------------------------------
```

$$-\nabla \cdot \nabla \Phi = \rho$$

$$-\int_V \nabla \cdot \nabla \Phi \, \varphi dV = \int_V \rho \, \varphi dV$$

$$-\int_V \nabla \cdot (\nabla \Phi \, \varphi) \, dV + \int_V \nabla \Phi \cdot \nabla \varphi \, dV = \int_V \rho \, \varphi \, dV$$

$$-\oint_{\partial V} \underbrace{(\nabla \Phi \cdot n)}_{=q_n} \varphi \, dV + \int_V \nabla \Phi \cdot \nabla \varphi \, dV = \int_V \rho \, \varphi \, dV$$

Poisson equation: source term, examples

Constant source:

```
Source = 1.0
```

Source dependeing piecewise linear on x:

```
Source = Variable Coordinate 1
    Real
      0.0 0.0
      1.0 3.0
      2.0 4.0
    End
```

Source depending on x and y:

```
Source = Variable Coordinate
  Real MATC ”sin(2*pi*tx(0))*cos(2*pi(tx(1))”
```

Source depending on anything

```
Source = Variable Coordinate 1
  Procedure ”Source” ”MySource”
```

Poisson equation: ElmerGUI menus

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE edf>
<edf version="1.0" >
  <PDE Name="Poisson" >
    <Name>Poisson</Name>

    <BodyForce>
    <Parameter Widget="Label" > <Name> Properties </Name> </Parameter>
       <Parameter Widget="Edit" >
         <Name> Source </Name>
         <Type> String </Type>
         <Whatis> Give the source term. </Whatis>
      </Parameter>
    </BodyForce>

    <Solver>
      <Parameter Widget="Edit" >
        <Name> Procedure </Name>
        <DefaultValue> "Poisosn" "PoissonSolver" </DefaultValue>
      </Parameter>
     <Parameter Widget="Edit">
        <Name> Variable </Name>
        <DefaultValue> Potential</DefaultValue>
      </Parameter>
     </Solver>

    <BoundaryCondition>
      <Parameter Widget="Label" > <Name> Dirichlet conditions </Name> </Parameter>
      <Parameter  Widget="Edit">
        <Name> Potential </Name>
        <Whatis> Give potential value for this boundary. </Whatis>
      </Parameter>
    </BoundaryCondition>
  </PDE>
</edf>
```

Development tools for ElmerSolver

- Basic use
  - Editor (emacs, vi, notepad++, jEdit,...)
  - elmerf90 script
- Advanced
  - Editor
  - git client
  - Compiler suite (gfortran, ifort, pathf90, pgf90,...)

  - Documentation tools (Doxygen, LaTeX)
  - Debugger (gdb)
  - Profiling tools
  - ...

Elmer – some best practices

- Use version control when possible
  - If the code is left to your own local disk, you might as well not write it at all
  - Never fork! (user base of 1000's)
- Always make a consistency test for a new feature
  - Always be backward compatible
  - If not, implement a warning in the code
- Maximize the level of abstraction
  - Essential for multi-physics software
  - E.g. any number of physical equations,
    any number of computational meshes,
    any number of physical or numerical parameters – without the need for recompilation