LARA Tutorial

A DSL-based Approach for Cross Layer Programming: Monitoring, Adaptivity and Tuning

Tiago Carvalho, Pedro Pinto, João Bispo, Ricardo Nobre, Luís Reis, and João M.P. Cardoso

University of Porto, FEUP, Porto, Portugal

April 20th, 2016

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

ANTAREX

SPeCS
Special-Purpose Computing Systems, languages and tools

Outline

- Tutorial Program
- Introduction:
  - Motivation
  - Our Approach
  - The LARA Approach
  - The Dynamic Extension of LARA
  - ANTAREX and LARA
  - Conclusions

Tutorial Program

- **Session A [14:00-15:30]:**
  - João M.P. Cardoso, *Introduction to the LARA Language and its use in the context of Toolchains and Design Flows* [20 min]
  - Pedro Pinto, Tiago Carvalho, *LARA for Programming Code Characteristics and Metrics* [20 min]
  - Pedro Pinto, Tiago Carvalho, *LARA for Programming Code Instrumentation Strategies* [25 min]
  - Ricardo Nobre, Pedro Pinto, *LARA for Programming Strategies for Code Transformations and Optimizations* [25 min]
- **Session B [16:00-17:30]:**
  - João Bispo, Luís Reis, *LARA in the context of a MATLAB/Octave to C/OpenCL Compiler* [45 min]
  - Tiago Carvalho, Ricardo Nobre, *LARA in the context of a Java to Java Compiler* [15 min]
  - Tiago Carvalho, Pedro Pinto, *LARA for Programming Strategies for Runtime Adaptivity* [15 min]
  - João Bispo, João M.P. Cardoso, *Plans for LARA in the context of the ANTAREX Project* [15 min]

Motivation

- Lack of support/mechanisms to
  - Specify strategies for code instrumentation and synthesis/compiler optimizations
  - Fully explore compiler optimizations
  - Apply the most suitable compiler sequence according to code and target architecture
  - Control tools in more advanced ways than the ones using pragmas/directives/switches
- No unified view of tool flows

Motivation

- Timing concerns...
- Performance concerns...
- Other concerns like the ones addressed by pragmas, code using #ifdef, etc.

```
void multigridsolver() {
#ifdef embedded_timing
  XTime_GetTime(&timeStamp_206983496);
  printf("TS-before 'gridsInit': %llu\n",
          timeStamp_206983496);
#endif
#ifdef pc_timing
  { hTimer t ("t1");
#endif
  gridsInit(&globalMap.m_obst,&globalMap.m_obst_div2,
            &globalMap.m_ip_brd2);
#ifdef embedded_timing
  XTime_GetTime(&timeStamp_206983496);
  printf("TS-after 'gridsInit': %llu\n",   timeStamp_206983496);
#endif
#ifdef pc_timing
  }
  printf("TS-after 'gridsInit': %llu\n",   hTimer::elapsed("t1");
#endif

}
```

timing for MicroBlaze

timing for PC

timing for MicroBlaze

timing for PC

Motivation

- Concerns tend to need code modifications, use of directives, etc.
  - Tangling and scattering
  - Code maintenance and evolution
- Concerns related to code transformations and compiler optimizations for:
  - Performance, Power, Energy
  - Parallelism, Concurrency
  - Monitoring, Test, Debug
  - Safety, Security
  - Targeting hardware accelerators, multicore and manycore architectures
  - Different tool flows

Existence of many crosscutting concerns!

Example of Concerns

HARRIS - Harris corner detector
Source code from:
http://slazebni.cs.illinois.edu/spring16/harris.m

```
function [cim, r, c] = harris(im, sigma, thresh, radius, disp)
    error(nargchk(2,5,nargin));

    dx = [-1 0 1; -1 0 1; -1 0 1]; % Derivative masks
    dy = dx';
    Ix = conv2(im, dx, 'same');    % Image derivatives
    Iy = conv2(im, dy, 'same');
    g = fspecial('gaussian',max(1,fix(6*sigma)), sigma);

    Ix2 = conv2(Ix.^2, g, 'same'); % Smoothed squared image derivatives
    Iy2 = conv2(Iy.^2, g, 'same');
    Ixy = conv2(Ix.*Iy, g, 'same');
    cim = (Ix2.*Iy2 - Ixy.^2)./(Ix2 + Iy2 + eps); % Harris corner measure
    if nargin > 2 % We should perform nonmaximal suppression and threshold
      sze = 2*radius+1;                   % Size of mask.
      mx = ordfilt2(cim,sze^2,ones(sze)); % Grey-scale dilate.
      cim = (cim==mx)&(cim>thresh);       % Find maxima.

      [r,c] = find(cim);                  % Find row,col coords.

      if nargin==5 & disp      % overlay corners on original image
          figure, imagesc(im), axis image, colormap(gray), hold on
          plot(c,r,'ys'), title('corners detected');
      end
    else  % leave cim as a corner strength image and make r and c empty.
      r = []; c = [];
    end
```

Code Tangling

Our Approach

- Aspectization of concerns
  - Concerns are specified as aspects using a domain specific language (DSL)
- A separation of concerns
  - Code not polluted, not intermingled, not duplicated…
  - Possibility to extend/enhance analysis and verification (by considering the well defined semantic and directly exposed behavior)
  - More opportunities for mapping and for code generation

LARA

Kiczales et al, Aspect-Oriented Programming. ECOOP'97

Hürsch&Lopes, Separation of Concerns, Northeastern University TR, 1995.

Specialization via Aspectization

```
function [cim, r, c] = harris(im, sigma, thresh, radius)

    dx = [-1 0 1; -1 0 1; -1 0 1]; % Derivative masks
    dy = dx';
    Ix = conv2(im, dx, 'same');    % Image derivatives
    Iy = conv2(im, dy, 'same');
    g = fspecial('gaussian',max(1,fix(6*sigma)), sigma);

    Ix2 = conv2(Ix.^2, g, 'same'); % Smoothed squared image derivatives
    Iy2 = conv2(Iy.^2, g, 'same');
    Ixy = conv2(Ix.*Iy, g, 'same');

    cim = (Ix2.*Iy2 - Ixy.^2)./(Ix2 + Iy2 + eps); % Harris corner measure

    sze = 2*radius+1;                   % Size of mask.
    mx = ordfilt2(cim,sze^2,ones(sze)); % Grey-scale dilate.
    cim = (cim==mx)&(cim>thresh);       % Find maxima.

    [r,c] = find(cim);                  % Find row,col coords.
```

Optimized version will provide dy as dx is a matrix of constants

specialized version will consider a specific sigma (1, 2, or 3) and will provide the required matrix g

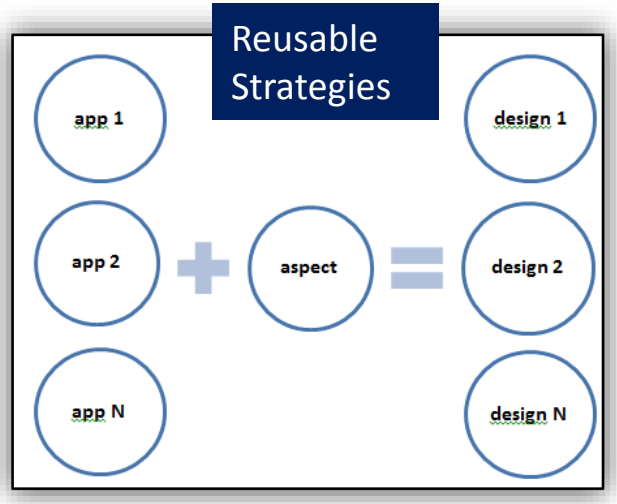
specialized version will consider the conv2 version considering 'same'

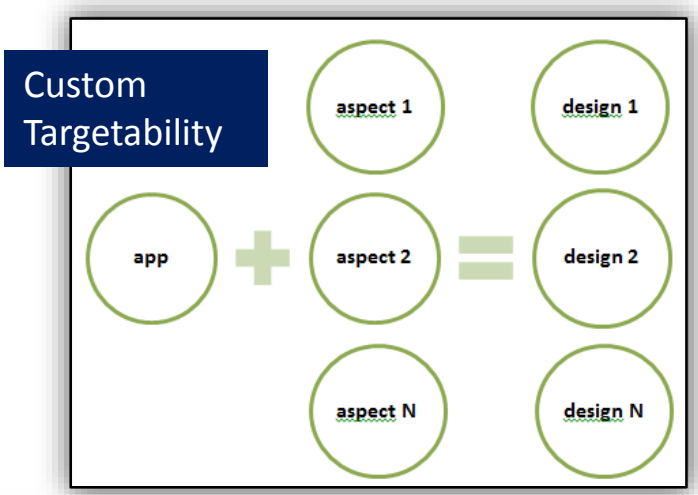
```
 if nargin > 2 % We should perform nonmaximal suppression and threshold
    else  % leave cim as a corner strength image and make r and c empty.
      r = []; c = [];
    end
```

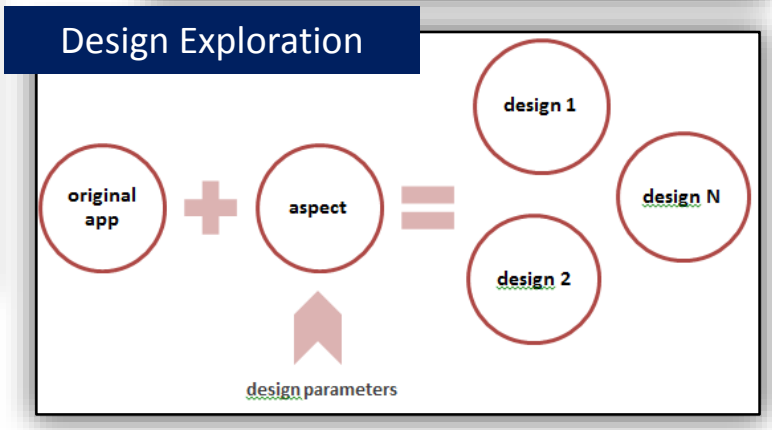
specialized version will consider ordfilt2 version w/o the 3rd parameter and specialized to a ones matrix as 3rd argument

LARA-based Approach

- Design Benefits

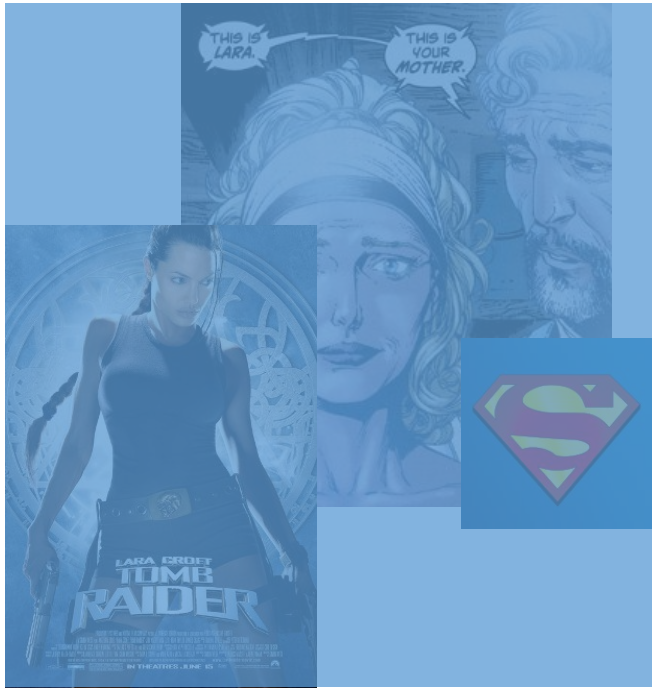






From REFLECT project

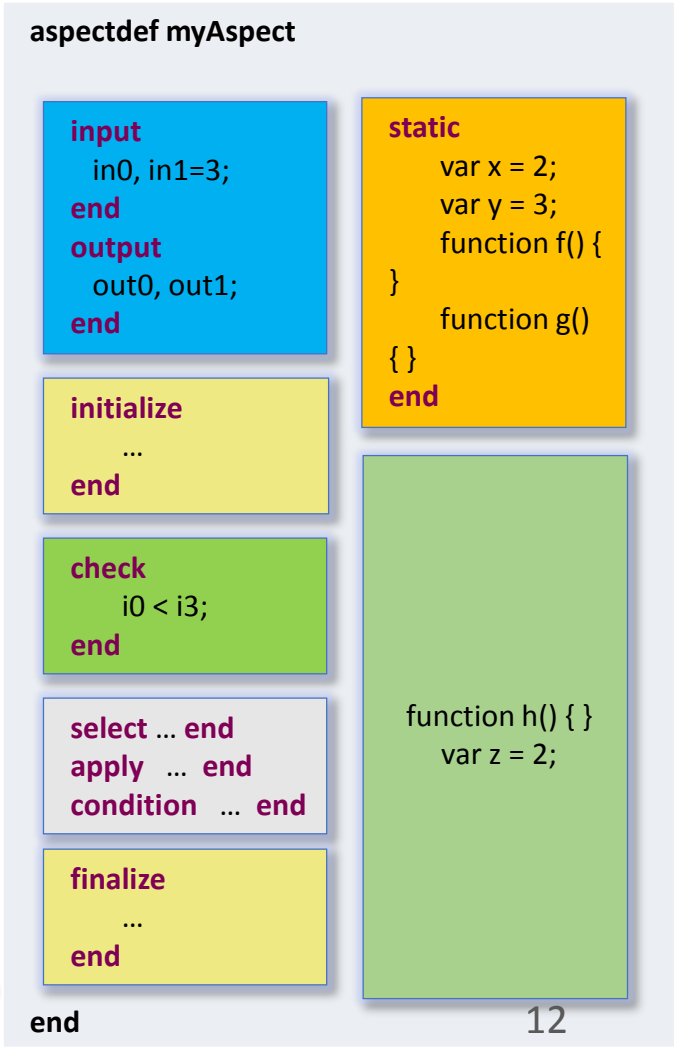
What is and What is not LARA?



- LARA is a Domain Specific Aspect Language
- It complements the application code
- It can be used to guide compilers and other tools in the toolflow

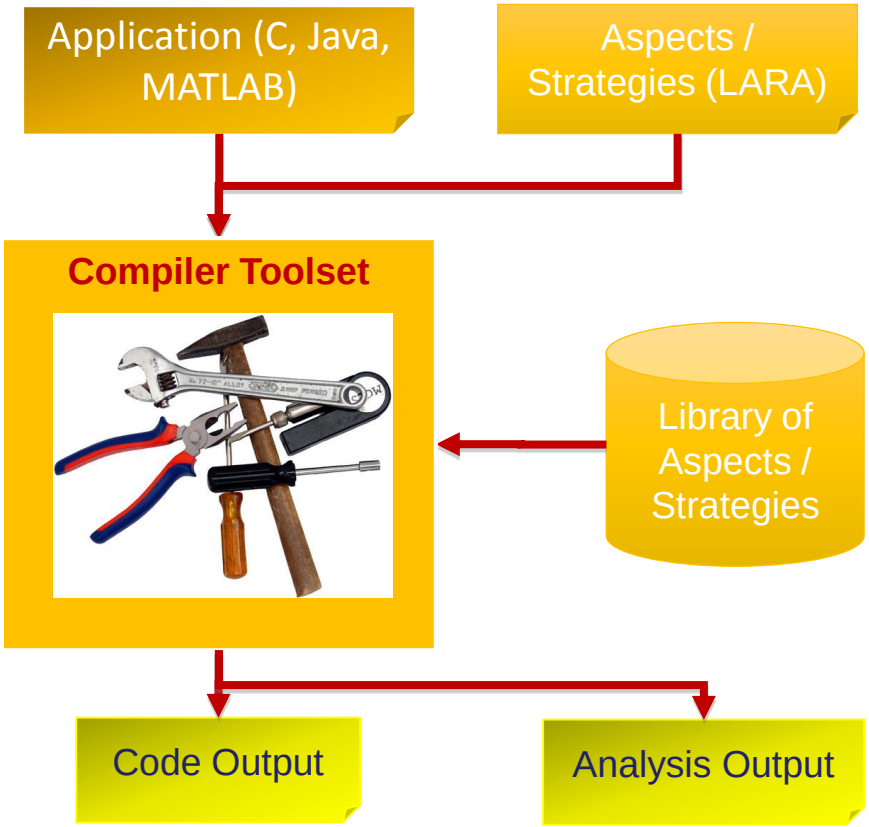
The LARA Language

- Secondary concerns detached from application logic code
- Useful to program strategies for instrumentation and synthesis/compiler optimizations
- Fully explore compiler optimizations and optimization sequences, according to code and target architectures
- Provides an unified view and DSE mechanisms



J. M.P. Cardoso, T. Carvalho, J. G. de F. Coutinho, W. Luk, R. Nobre, P. C. Diniz, Z. Petrov, "**LARA: An Aspect-Oriented Programming Language for Embedded Systems,"** in *Int'l Conf. on Aspect-Oriented Software Development (AOSD'12)*, Potsdam, Germany, March 25-30, 2012.

LARA-based Tool Flow



LARA-based Tool Flow

Application

```
void filter_subband(float
z[512], float s[32], float
m[32][64]) {
  ...
  for (i=0;i<32;i++) {
     s[i]= 0;
     for (j=0;j<64;j++) {
        s[i] += m[i][j] * y[j];
     }
  }
  ...
```

Compiler Toolset

Aspects and Strategies

```
aspectdef monitor1
  select function.var{"s"} end
  apply
    insert.after %{if([[$var.usage]] >= 10)
       printf("Warning: value >= 10!\n");} %
  end
  condition $var.is_write end
end
```

Program elements

Advices (actions)

Condition

Code Output

```
...
  for (i=0;i<32;i++) {
     s[i]= 0;
     if(s[i] >= 10)  printf("Warning: value >= 10!\n");
     for (j=0;j<64;j++) {
        s[i] += m[i][j] * y[j];
        if(s[i] >= 10)  printf("Warning: value >= 10!\n");
     }
  }
...
```

LARA Action: Code Instrumentation

LARA Main Features

- Declarative select-apply clauses
- Composition of strategies based on other strategies
- Modularity and resuse based on calling aspects and using parameters

```
select function end
apply
    …
end
```

```
apply
  call loopunroll();
  call timing();
end
```

```
apply
  call loopunroll(8, 64);
end
```

LARA Main Apply Actions

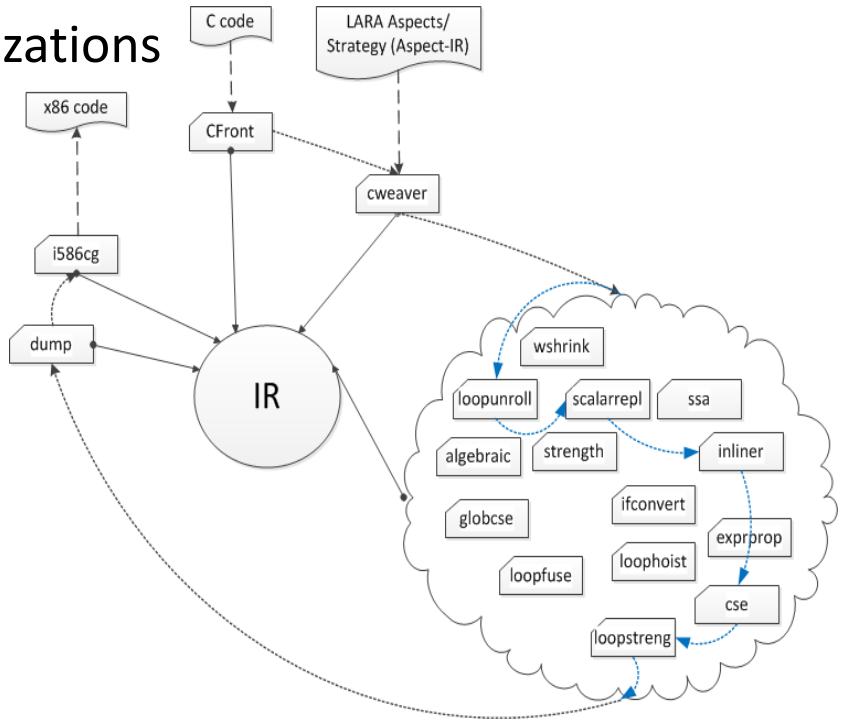
- Insert before|after|replace
  - For injecting code in input application source code
- Exec
  - For executing a compiler action (e.g., loop unrolling)
- Def
  - For defining the value of a property (e.g., the type of a variable)
- Run
  - For executing an external tool

```
insert before ’code to inject’;
$call.insert before ’code to inject’;

exec Unroll();
$loop.exec Unroll();

def type=“float”;
$var.def  type=“float”;

run(“analyze”, filename);
```

LARA Actions: Seq. of Compiler Opt.

- Specify sequences of compiler optimizations

```
aspectdef optimizationseq
    select function end
    apply
        exec loopinvariant();
        exec loopscalar();
        exec dismemun();
        exec loopstrength();
        exec strength();
        exec looprev();
        exec lowerboolval();
        exec loopbcount();
    end
end
```



LARA Action: executing compiler stages

Code Transformation Strategy: Loop Unrolling

```
aspectdef LoopUnroll
    select loop end
    apply
        if($loop.num_iterations <= 32) {
            $loop.exec Unroll(0);
        } else {
            $loop.exec Unroll(2);
        }
    end
    condition
        $loop.is_innermost &&
        $loop.type=="for"
    end
end
```

- Selects every loop in the program
- Loops with less than 32 iterations:
  - Are fully unrolled
  - Uses a factor of 2 otherwise
- Applies transformation if loop:
  - is innermost
  - is a FOR loop
- More sophisticated analyses/strategies are possible:
  - Using attributes
  - JavaScript code

LARA: “receipts” (recepies) for compiler optimizations

LARA Strategies

- Recursively unroll loops fully or 2×, depending on their charecteristics

Input Program

```
…
for (i=0;i<64;i++) {
  y[i] = 0;
  for (j=0;j<8;j++)
    y[i] += z[i+64*j];
}
for (i=0;i<32;i++) {
  s[i] = 0;
  for (j=0;j<64;j++)
    s[i] += m[i*32 +j] * y[j];
}
…
```

```
…
for (i=0;i<64;i+=2) {
  y1 = z[i]; …
  y1 += z[i+64*7];
  y[i] = y1;
  y1 = z[i+1]; …
  y1 += z[i+1+64*7];
  y[i+1] = y1;
}
for (i=0;i<32;i+=2) {
  s1 = 0;
  for (j=0;j<64;j+=2) {
    s1 += m[i*32+j]*y[j];
    s1 += m[i*32+j+1]*y[j+1];
  }
  s[i] = s1;
  …
}
…
```

Strategies

```
aspectdef Strategy
  input fn="f1" end
  select function{name==fn} end
  apply
    do { call loopunroll(8, 64); } while($function.changed);
  end
end
```

```
aspectdef loopunroll
  input niter1=10, niter2=20 end
  select loop{type=="for"} end
  apply
    exec loopscalar;
    if($loop.num_iter <= niter1) {
      exec loopunroll(k:"full");
    } else if($loop.num_iter <= niter2) {
      exec loopunroll(k:2); $loop.already="true";
    }
  end
  condition
    !$loop.already && $loop.is_innermost &&
    $loop.numIterIsConstant
  end
end
```

Related Work

- Quering the source of software programs:
  - OMEGA [LintonSDE'1984]
    - Provides mechanisms for accessing and displaying the information in a large software system
    - Used QUEL, a relational database query language
  - The C Information Abstraction System [ChenIEEE-TSE'1990]
- Aspect-Oriented Programming (AOP)
  - AspectJ [Kiczales, et al., *ECOOP'97*]
  - AspectC++ [Spinczyk, TOOLS-Pacific'2002]
  - Functional queries and composable queries for pointcut designators [Eichberg et al., PQL'2004].
  - Eos and Eos-T [RajanAOSD'05] [RajanSIGSOFT-SEN'2003]
- Our work on
  - [2007-2011] **AMADEUS** (Aspects and compiler optimizations for matlab system development) , FCT Project
  - [2010-2012] **REFLECT** (Rendering FPGAs to Multi-Core Embedded Computing) , FP7 Project

Tools used in the Tutorial

- MANET
  - C to C Compiler based on Cetus + LARA
  - Demo version: http://specs.fe.up.pt/tools/manet/
- KADABRA
  - Java to Java Compiler based on Spoon + LARA
  - Demo version: http://specs.fe.up.pt/tools/kadabra
- MATISSE
  - MATLAB to C/OpenCL Compiler + LARA
  - Demo version: http://specs.fe.up.pt/tools/matisse

Thank you! Questions?

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

ANTAREX 10^{18}

SPeCS
Special-Purpose Computing Systems, languages and tools