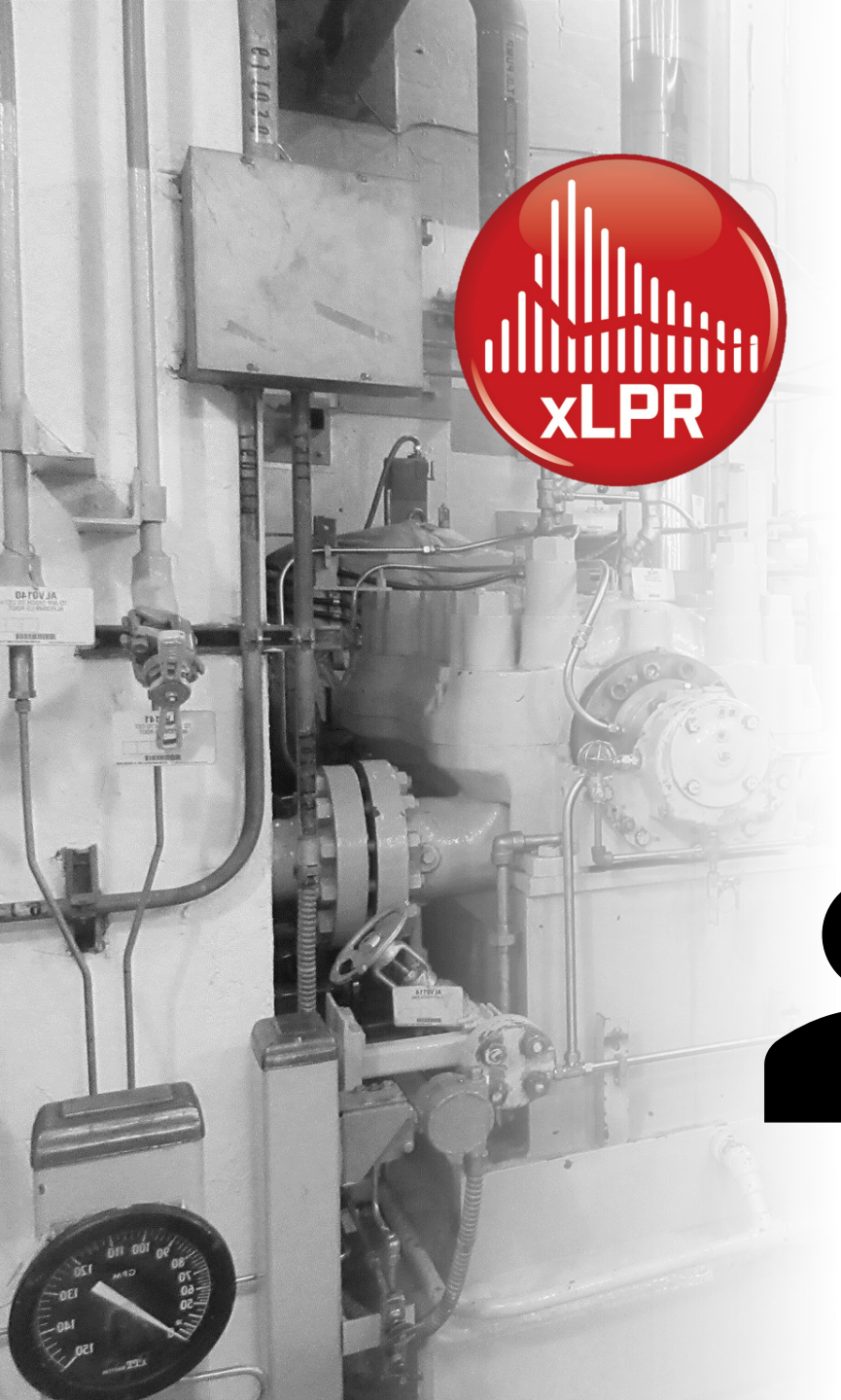
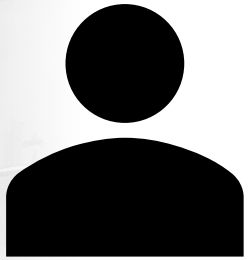


Extremely Low Probability of Rupture Code

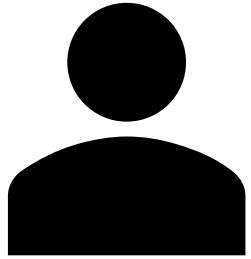
NRC Update 2025

Christopher Nellis



U.S. Nuclear Regulatory Commission
Office of Nuclear Regulatory Research
Division of Engineering
Reactor Engineering Branch

Hector Mendoza



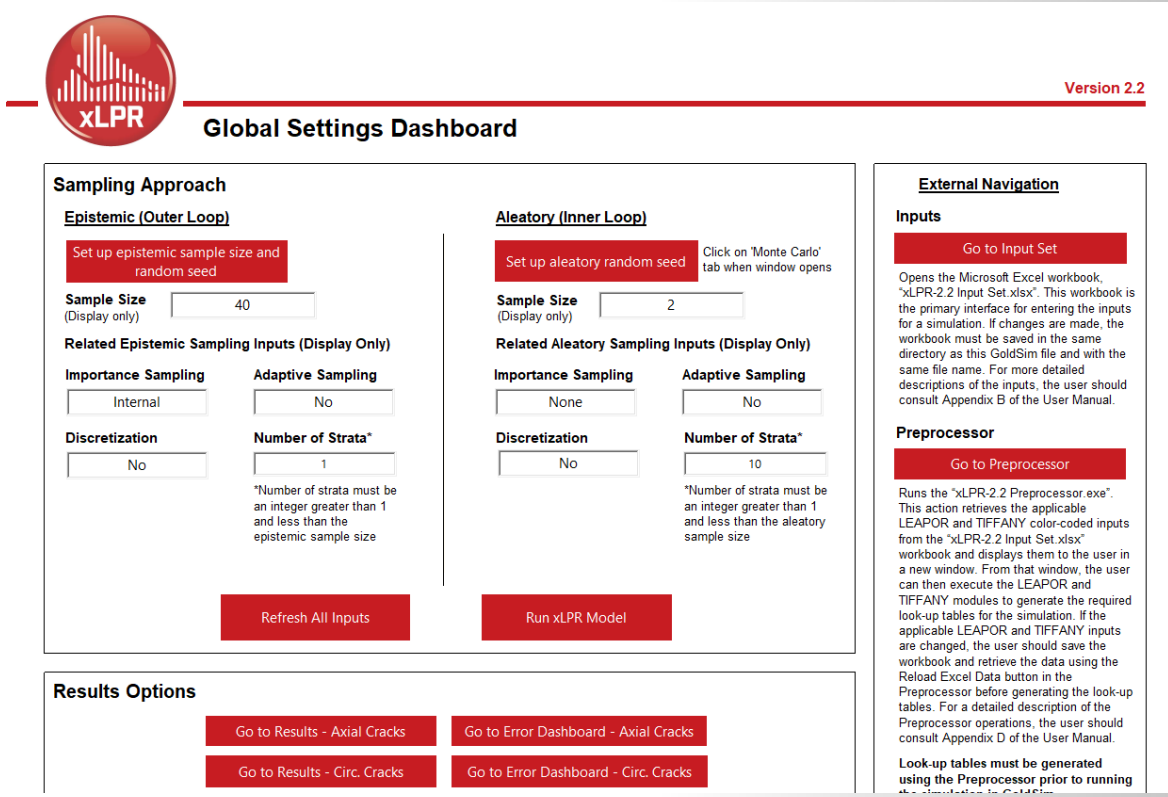
U.S. Nuclear Regulatory Commission
Office of Nuclear Regulatory Research
Division of Engineering
Reactor Engineering Branch

Agenda

- Background
- xLPR Version 3.0
  - What is the xLPR Framework?
  - Why are we replacing the Framework?
- Project Objectives
- Project Timeline
  - Language Selection
  - Current Status
- Future Plans
  - xLPR Development for CISCC

Extremely Low Probability of Rupture (xLPR) Code Overview

- Probabilistic fracture mechanics (PFM) software tool for nuclear power plant piping integrity risk analysis
- Joint effort by the NRC's Office of Nuclear Regulatory Research and the Electric Power Research Institute (EPRI), now in second major version (v2.3 soon to be v2.4) with a third major version being developed (v3.0)
- Capable of modeling the effects of stress-corrosion cracking (SCC) and thermal fatigue
- Used by NRC and EPRI staff and contractors to risk-inform industrywide emerging piping integrity issues via probabilistic approaches



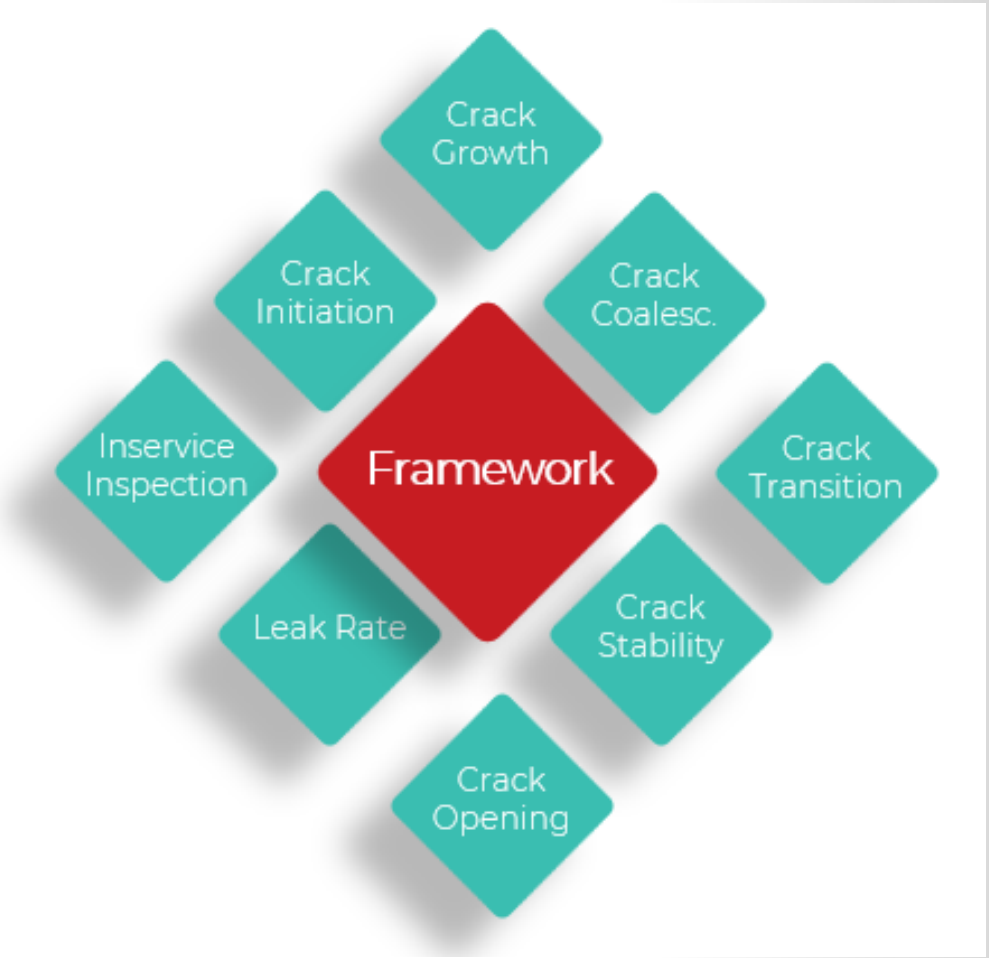
xLPR Modular Design

Framework

- Reads input parameters
- Performs sampling on probabilistic parameters
- Interfaces with FORTRAN modules containing physical models
- Collects module outputs

Modular Design

- Each FORTRAN module contains the physical models behind xLPR
  - Crack Initiation, Crack Growth, Inservice Inspection, etc
- Each module developed by technical experts
- Allows flexibility of future development



Known Framework Needs

Adaptability

- Move away from requiring access to Commercial Off-the-Shelf (COTS) software
- Reduce the learning curve to making modifications
- Replace certain features that are proprietary and locked from users

Performance

- Currently there is limited parallelization capacity
  - 4 cores for free GoldSim version
  - 10 cores for subscription GoldSim
- Memory issues with large sample sizes
- 32-bit COTS application limits improvements to performance

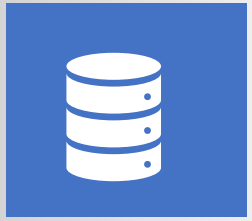
xLPR 3.0 Project Goals

Goal #01

Increase flexibility by gaining full control of the source code.

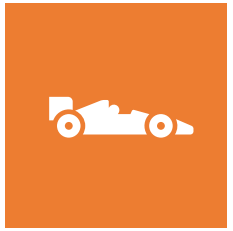
Goal #02

Increase maintainability by leveraging agile processes and modern software development tools.



Goal #03

Increase capacity by enabling access to more memory resources and minimizing repetitive human operator tasks.



Goal #04

Increase performance through increased parallelization on large virtual machines.

xLPR 3.0

Process for Developing the New Framework

01 Programming Language Evaluations

Candidates: Python, C++, Fortran, Java, and Go

02 Establish Requirements and Software Architecture

Perform MoSCoW analysis for requirements (Must Have, Should Have, Could Have, Won't Have) and establish the new, high-level software architecture

03 Development Infrastructure

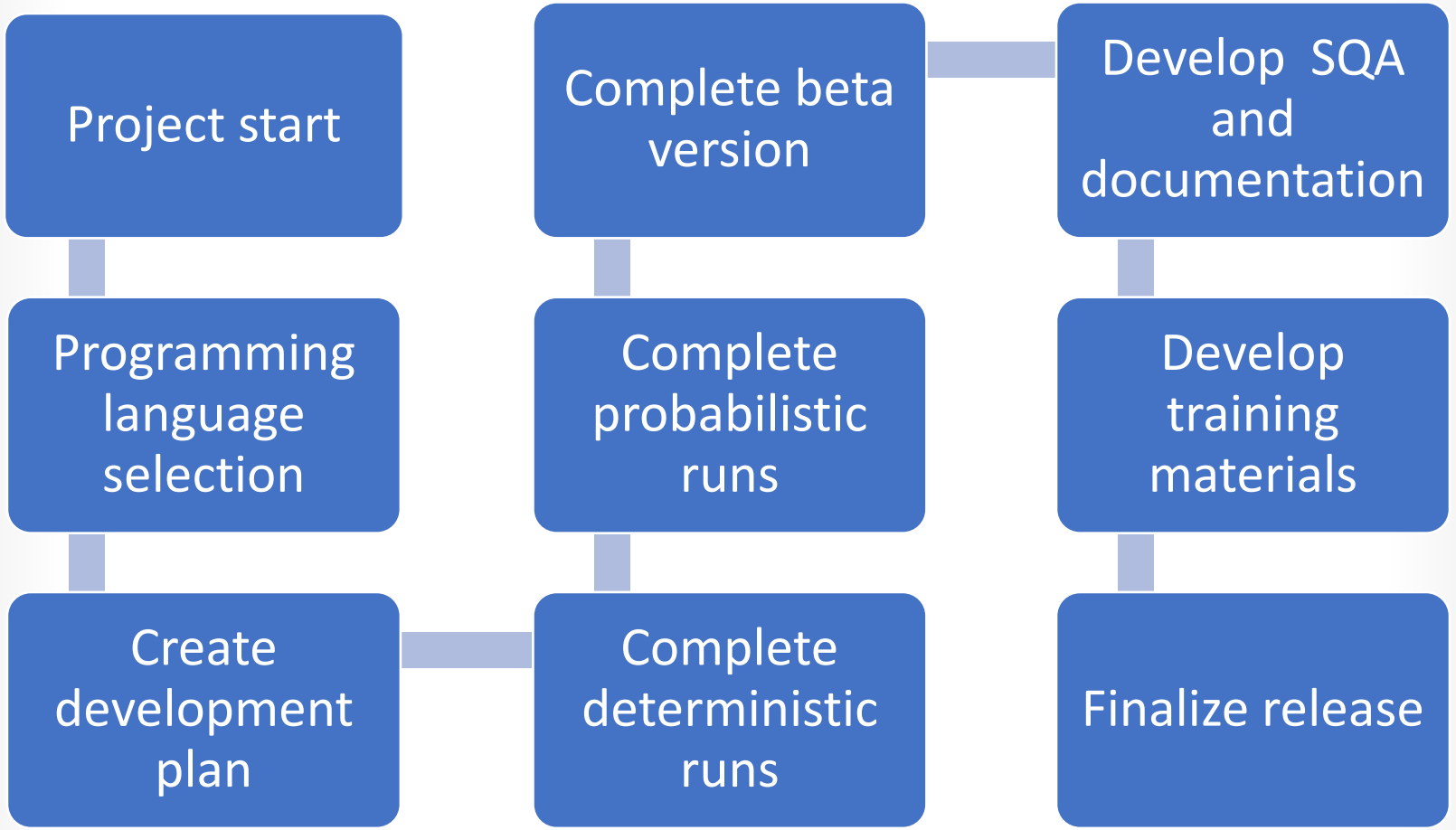
Top-down GitHub integration, doorstop for requirements management, and automated testing tools

04 Run Development Process

Establish product goals and supervise ongoing development

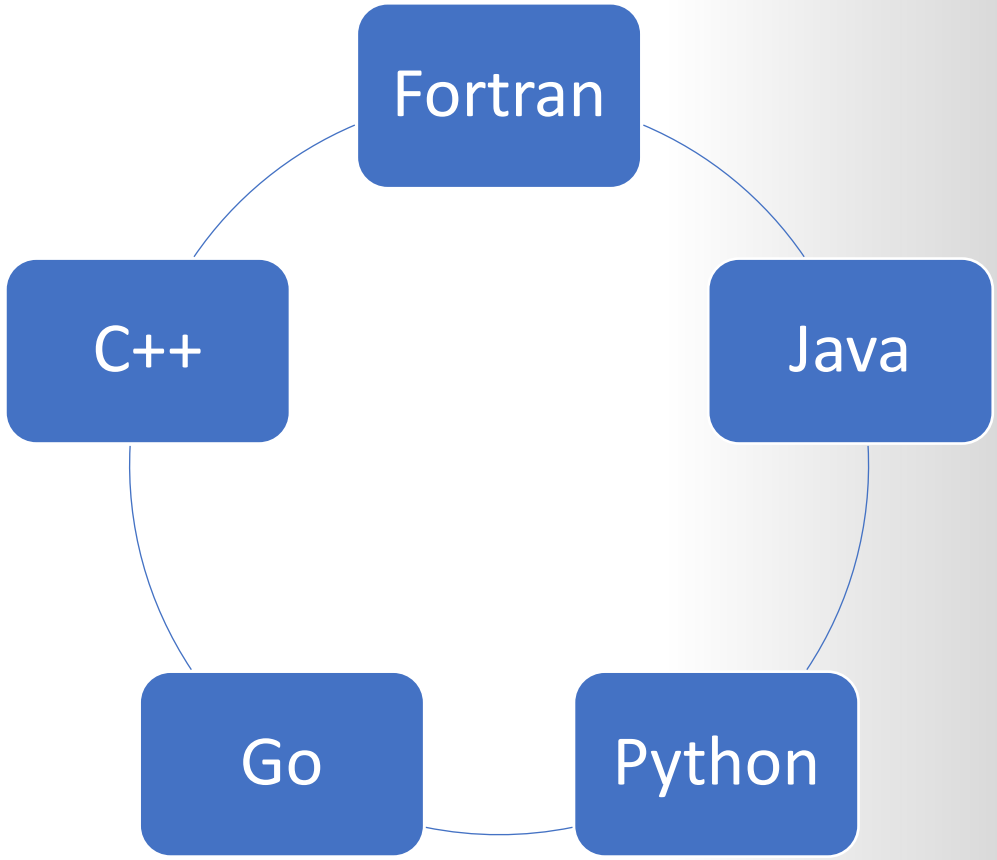
05 Software Release and Training

xLPR 3.0 Timeline



Language Recommendation

- Developed evaluation criteria for Framework programing language
- Consulted with development team and stakeholders to rank according criteria
- Conducted evaluations to make informed decision on Framework language



Evaluation Criteria

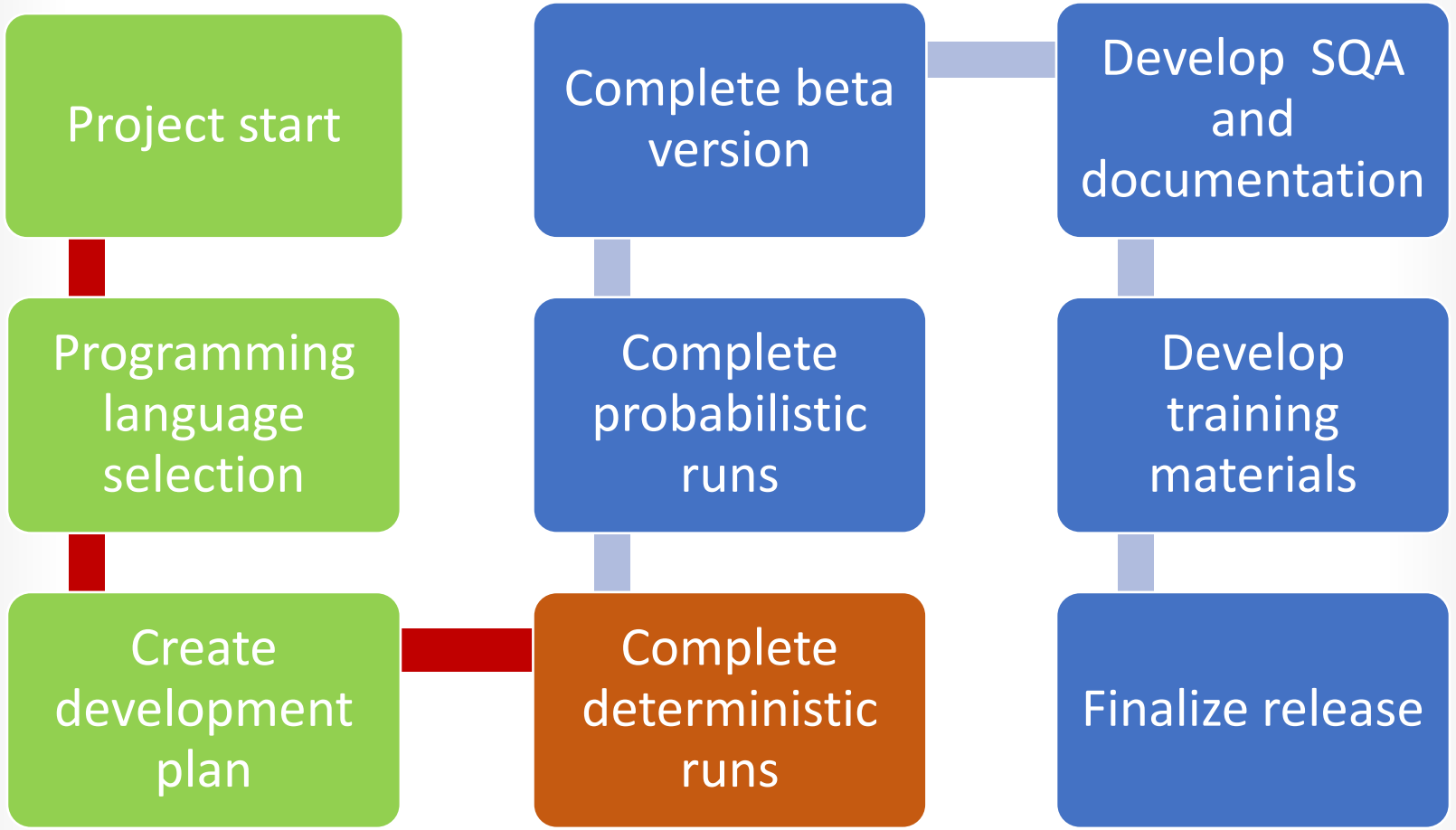
- Ranked Topics
  - Support for statistical functions
  - Support for visualization
  - Support for unit conversions
  - Ease of distributed computing
  - Object-Oriented design
  - Ease of parallelization
  - Computational speed
  - Handling of large data structures
  - Extensibility
  - Memory Requirements
  - Operating environment support
  - Memory safety
  - Fortran Interface
  - Excel Interface

- Ranked Topics
  - Excel Interface
  - Availability of online resources
  - Future NRC/EPRI Engineer Knowledge
  - Ease of initial development
  - Ease of code maintenance
  - Ease of learning/gain proficiency in language
  - NRC-EPRI coding team experience
  - Distribution of executable
  - Licensing of third-party libraries
  - Typing
  - Webhosting
  - Interfaces
  - Testing

Language Selection

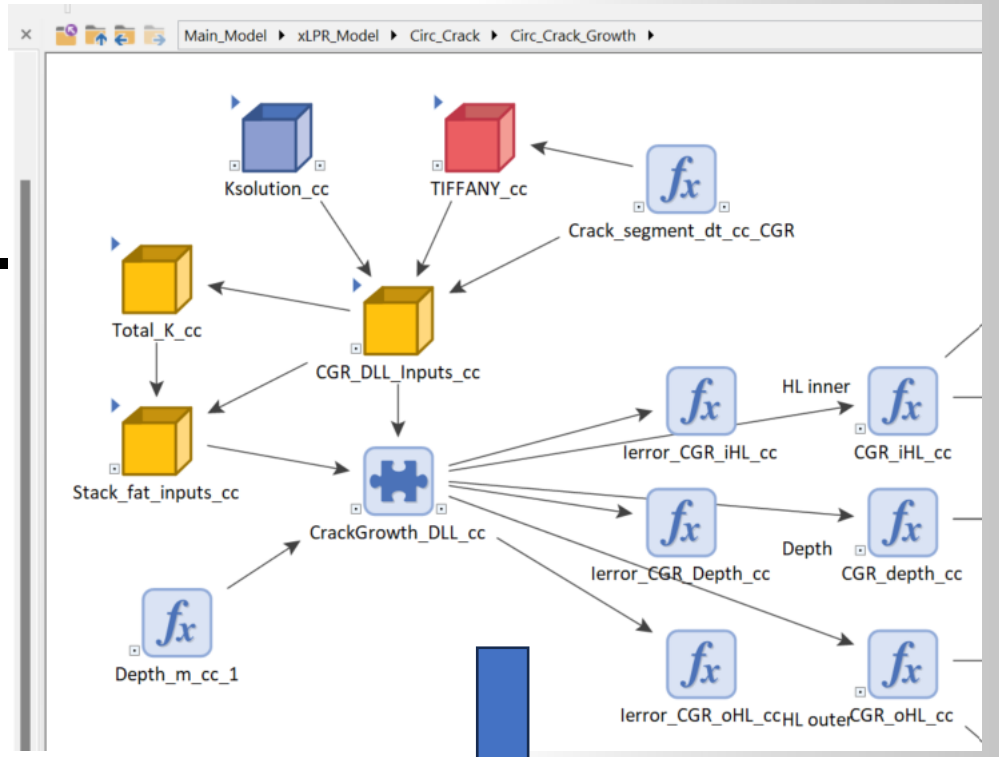
- Go has a syntax similar to C++ and Java and is simpler to learn
- Go eliminates aspects of C++ that are problematic and emphasizes features that improve maintainability
- The Go language is designed to limit dependencies that make code hard to modify
- Go was designed to simplify parallel computing and make threads more efficient
- Go compiles to an executable that incorporates Fortran routines and simplifies distribution

xLPR 3.0 Timeline



Framework Conversion

- 4000+ unique elements in the GoldSim framework need to be converted into Go code
- Create new interfaces between the Go Framework and the compiled FORTRAN DLLs
- Confirming consistency of outputs from DLLs compared to xLPR 2.3




```go
// GOLDSIM: CrackGrowth_DLL_cc; Pos 1309
//
// A [fc.FortranFnWrapper] that computes the crack growth of all initiated cracks.
type CrackGrowthCCcalc struct {
    name            string
    calculationType fc.FortranFnType
}

func (c CrackGrowthCCcalc) Name() string {
    return c.name
}

// Returns the [fc.FortranFnType] of this [fc.FortranFnWrapper]
func (c CrackGrowthCCcalc) FortranFnType() fc.FortranFnType {
    return c.calculationType
}

// Runs the growth subroutine from the CGR Fortran90 file.
func (c CrackGrowthCCcalc) DoStep(currentParams pc.ParamCollector) {
    status := C.int(0)
    outputs := make([]C.double, 211)
    inputArray := c.getInputs(currentParams)
    C.growth(&status, &inputArray[0], &outputs[0])
    c.storeOutputs(outputs, currentParams)
}
```

xLPR 3.0 Summary

- Selected Go programing language for Framework
- Proceeding through manual conversion of GoldSim elements into Go functions
- Anticipate completion in early 2026

The xLPR code is continuing to evolve as a high technology platform for nuclear systems and component integrity assessment.

xlpr@nrc.gov or xlpr@epri.com for further information

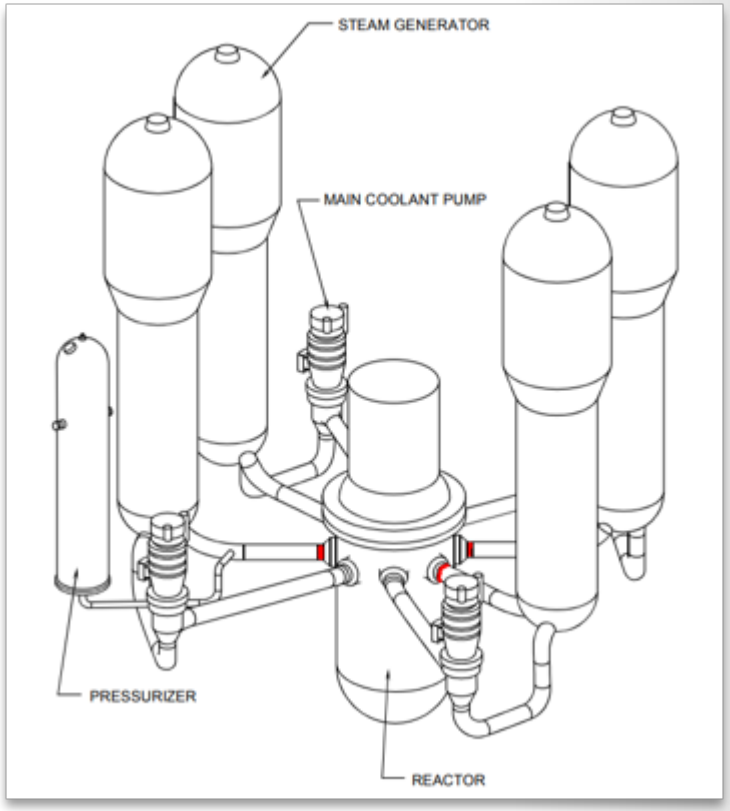
xLPR Use Cases

Latest NRC Use Cases

xLPR is used in support of NRR's LIC-504 analysis of the risk of the emergent French Stress Corrosion Cracking OE to the US PWR fleet (ML24162A131)

xLPR used to perform analyses in NRR review of Leak-before-Break (LBB) regulatory approach (ML21217A088 and ML22088A006)

- Developing PFM tool in xLPR to model CISCC of dry storage canisters in FY25
- Using the tool with site-specific conditions to assess CISCC, report in FY26



CISCC Modeling Research Objectives

Risk-inform aging of dry fuel storage by chloride-induced stress corrosion cracking (CISCC) of stainless-steel canisters

Build on previous work assessing uncertainties in modeling CISCC including key environmental and material parameters *ML24183A005*

Develop a probabilistic fracture mechanics (PFM) tool to model CISCC of stainless-steel canisters to risk-inform the impact of site-specific conditions as follow on work to RG 3.78

xLPR Development for CISCC

Additional modules and framework modification to model Chloride-induced stress corrosion cracking (CISCC) to xLPR

Developing CISCC **crack initiation** and **crack growth** modules

Supplementary modules for stress intensity factor and weld residual stress

Framework modification to support unique CISCC problem space

Summary

xLPR 3.0 anticipated for release in early 2026

Development of xLPR modules for CISCC expected to be complete in 2025 and in use for site specific analysis in 2026

xlpr@nrc.gov or xlpr@epri.com for further information