EUROPEAN SPALLATION SOURCE

Commissioning of software at ESS

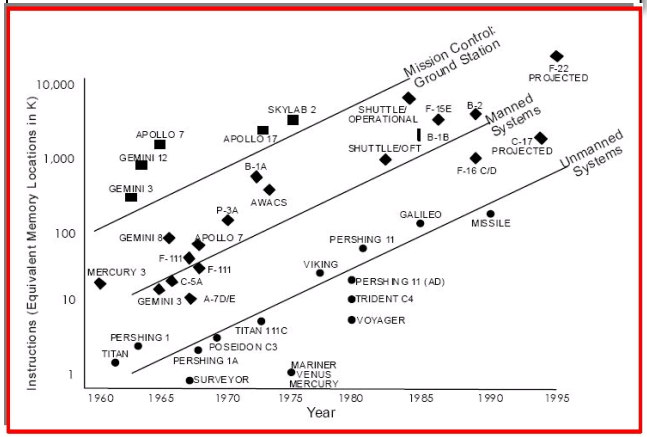
Henrik Carling

ESS/ICS
Date: 2017-06-13

Importance of software

- ESS is a technically challenging project
- Software is - as everywhere else in science and technology - becoming ever more important
- Software is also more and more involved in monitoring and controlling critical functions. We become dependant on software
- The integrated control system depends on large software systems, such as EPICS, CS Studio etc. to provide reliable control functions to ESS
- Software design, development, test and deployment are of great importance for the outcome of the project

```
#include <stdio.h>
#include <string.h>

char X[] =
  " ETIANMSURWDKGOHVF:L:PJBXCYZQ::54:3:::2&:+::::16=/:::(:7:::8:90"
  ":::::::::::?_::::\"::.::::@:::'::-:::::::::;!:):::::,::::::";

int j,w,h,f,u,d,g,e,n,i,l,a,r,p,c,m,o,t,s,Q=32,T[32],W[32],I=13000;

int main(int v, char** b) {

  for (i=0;v>0&&v<5&&(v*=t=fread(X,1,1,stdin));)

  for (v-1&&101==*1[b]?++g%1500?u+=(g&1^1)*(*X<0?-*X:*X):(f=e,e=w,w=d,d=u,
    m=(1 - h%2* 2  )*(d -f)/ (d<f  ?d|1 : 1|f) >  5?T[ h%Q] =o+l  , l=W[
    h++%Q]=o,o=0:m,o++,u=main(0,b)):(c=strrchr(X,~(Q&*X&*X/2)&*X)-X,j=255);
    v==1&&(c*j||main(*X-Q?8:24,b));j/=2)c<j?0:main(9+(c-j)/(j/2+1)%2*10,b);

  for (a=u=0;i<=Q*Q&&!v;j>7&&j<13?s=c*s+T[i/Q]/2,s/=++c:0)
    !(i++%Q)?a=c>a?u=s,c:a,c=s=0:0,j=T[i%Q],j=j?T[i/Q]*10/j:0,j*=(j<5)*2+1;

  for (r+=(h-r)/Q*Q;i<v/4*I;putchar((i/I<v%4)*i%2*85<<(i%176/88)),i++);

  for (g*=!!v;r+t!=h+1&&h>5&&!v;r+t!=h+1?(W[r%Q]>2*u||r==h?p=n=n>6?0:
    putchar(X[p-1+(1<<n)])&0:0,W[r++%Q]>6*u?putchar(Q):0):0)
    *X=Q,p=r%2?++n,2*p+(W[r++%Q]>2*u):p;

  return 0;
}
```



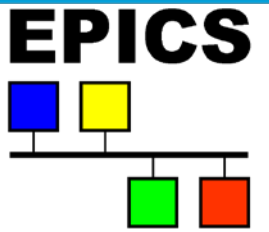
Source code: http://www.ioccc.org/
Statistics: http://sce2.umkc.edu/BIT/burrise/pl/introduction/

The pieces of the puzzle

- ESS will use many ready software packages developed by the scientific community, which are available as open-source products
- Only a small fraction of the software will be developed specifically by/for ESS
- However, all software must be installed, configured, staged, tested and deployed for production in a managed and reliable way
- All software products will eventually be integrated into the ICS

- EPICS
  - pvAccess
  - Channel finder
  - Control system studio
- OpenXAL
- Logbook
- Alarm system
- Save, Compare & Restore
- Archiving
- ESS configuration management suite
  - CCDB
  - CDB
  - Calibration service
  - IOC factory
  - Naming service
- Post Mortem service/application
- Diagnostics service/application

and more



The EPICS Archiver Appliance

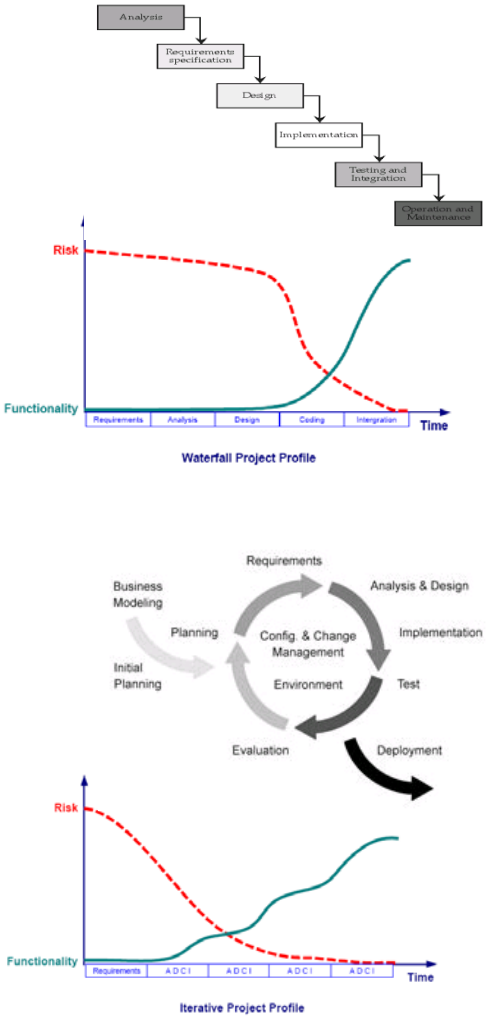
Modern software development methodology

- When is a software development project done?
  - *Never!*

- Software project requirements often change
- Software can be improved and augmented even after a product is launced or in production
  - This puts high demands on the software quality

- Traditional waterfall planning methodologies have been replaced with agile project methodologies

- One part of agile methodology is Continuous Integration
  - An important part is to develop software in small steps



Images: https://dandriani.wordpress.com/2012/09/06/41/

Principles of continuous integration

- With continuous integration, it is possible to continuously correct, improve and augment developed software avoiding ”big-bang” transistions of software deployment
- Through strict version control enabled by versioning software such as git, a fully traceable and managed release mechanism is enabled
- Continuous integration demands a focused and stringent investment in code quality through reviewing/gatekeeping and extensive testing
- The testing in continuous integration schemes is always automated to the highest possible extent

Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day.

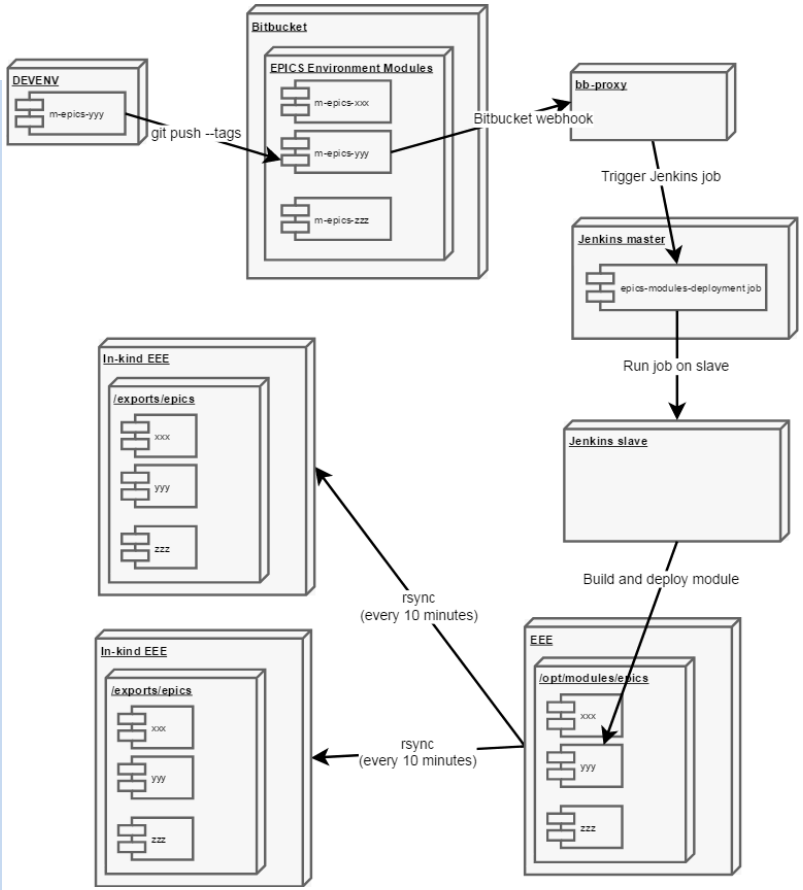
Each check-in is then verified by an automated build, allowing teams to detect problems early.

Source:
https://www.thoughtworks.com/continuous-integration

Continuous integration at ESS

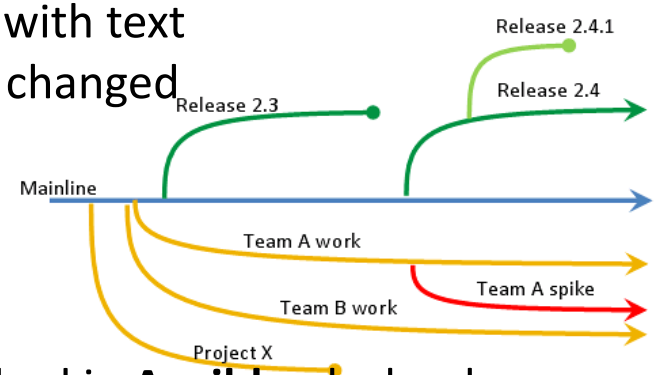
- A scheme has been devised for continuous integration at ESS

- Everyone commits every day (or multiple times a day)
- Maintain a fast build process
- Always run your commits locally and test your code before committing to the repository and trigger continuous integration.
- Wait for the feedback reports before moving on
- Never leave a broken build. You could block your colleagues working on the same project
- Always be prepared to revert to a Previous Version
- Do not comment out failing tests
- Take responsibility for the breaks that are the result of your code
- Use Test-Driven Development is possible



”Everything is code”!

- Also configuration itself needs to be tightly managed and versioned
- Version management systems and humans interact well with text
  - You can do differentials and easily identify what has changed
  - You can manage versions of the configuration
- ICS’ ambition is to make all ESS configuration textual
- All the configuration of ICS computer systems are described in **Ansible** playbooks
- Ansible allows ICS to describe all configuration as code (*IaC) and to keep such configuration in a version control system such as git
- This lets us version control the current configuration of ICS systems and therefore keep control of the updates, and even perform rollbacks



* Infrastructure as Code

The ESS EPICS Environment - EEE

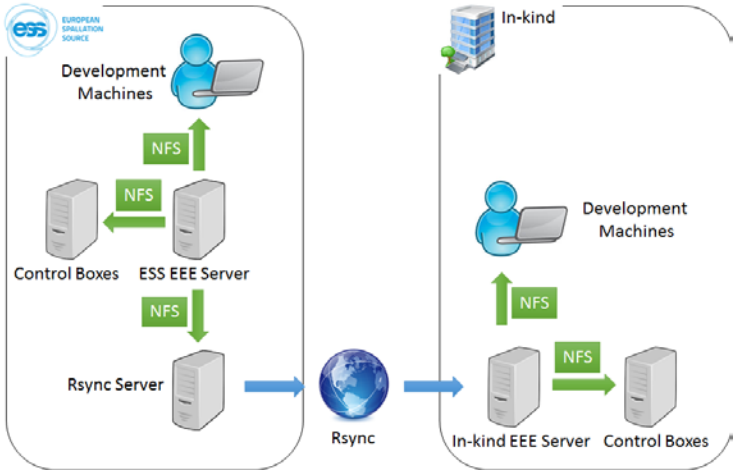
- ICS provides all the tools needed to build software for the commissioning and operation of ESS to internal developers as well as in-kind collaborators and external suppliers

- EPICS development environment is managed by a centralized system that is called the ESS EPICS Environment (EEE)

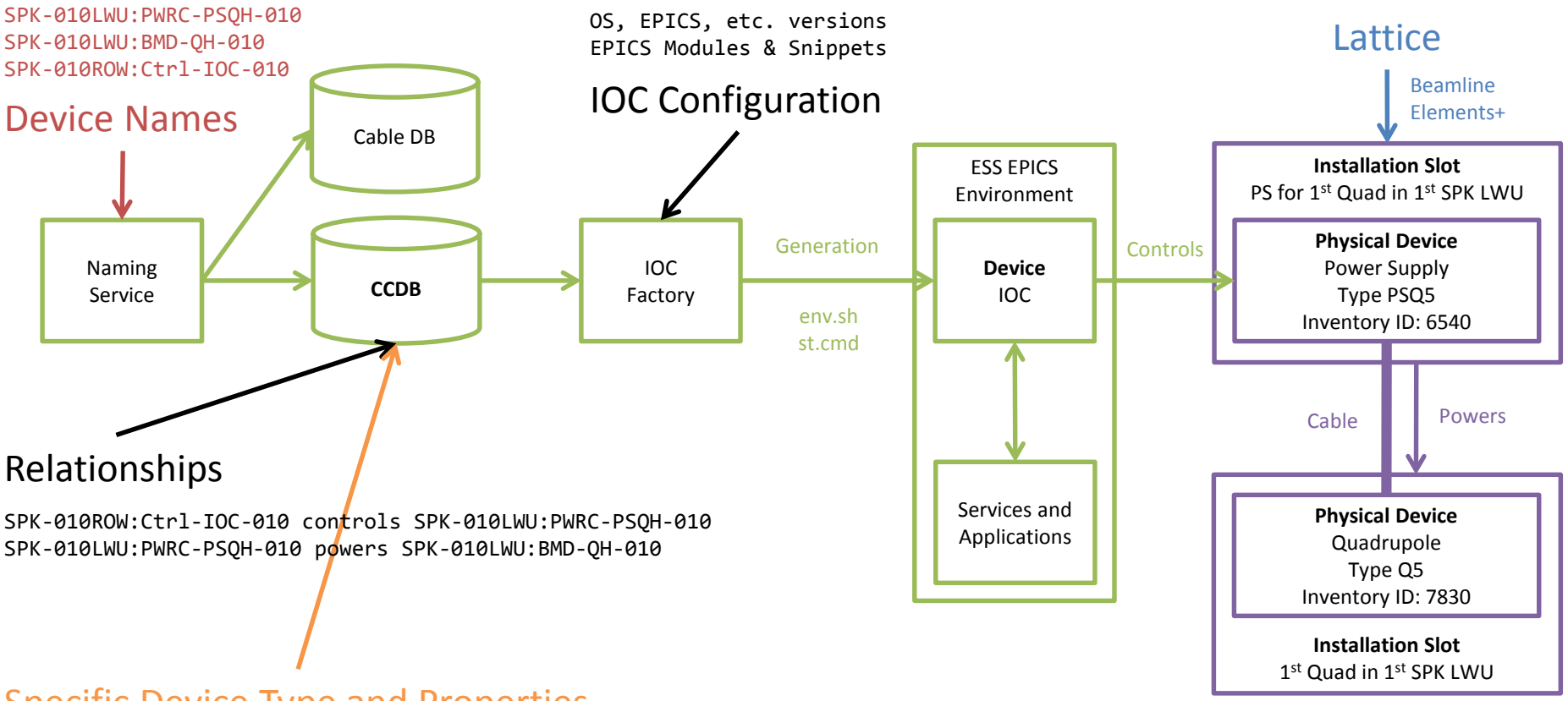
- When new functionality is added to the EEE server at ESS, it is made available not only to ESS but also to external developers

- This is achieved by periodic one-way synchronizing of in-kind EEE servers with the ESS centralized EEE server

- Synchronization is performed over the Internet using Rsync



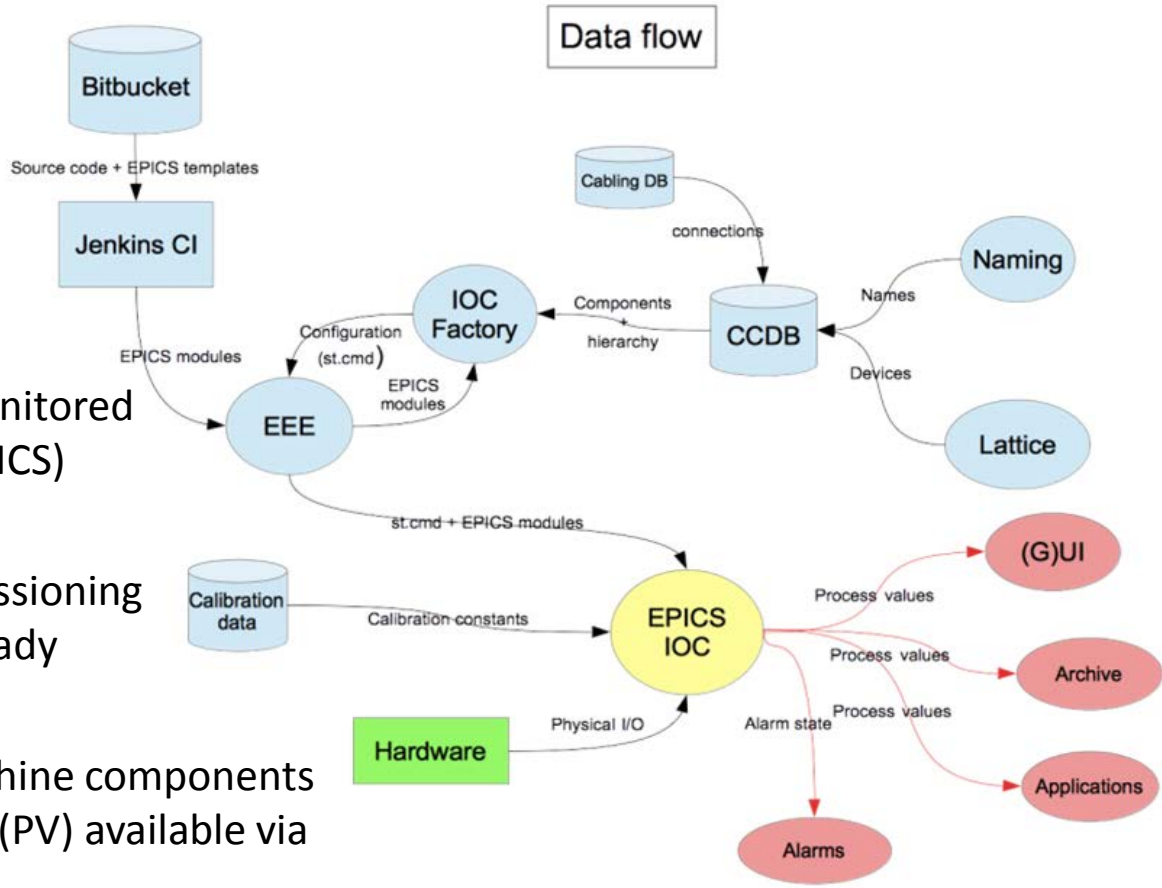
Controls Configuration Management Workflow



Overview by Iñigo Alonso

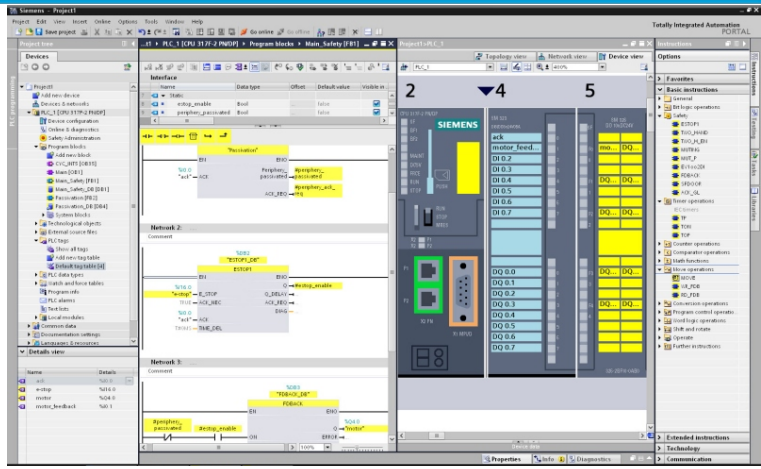
ICS configuration data flow

- ESS has thousands of components that need to be controlled and/or monitored from the Integrated Control System (ICS)
- Installation is starting, and for commissioning some services from ICS need to be ready
- To interact with the ICS services, Machine components need to make their Process Variables (PV) available via Input Output Controllers (IOC)



Industrial automation software

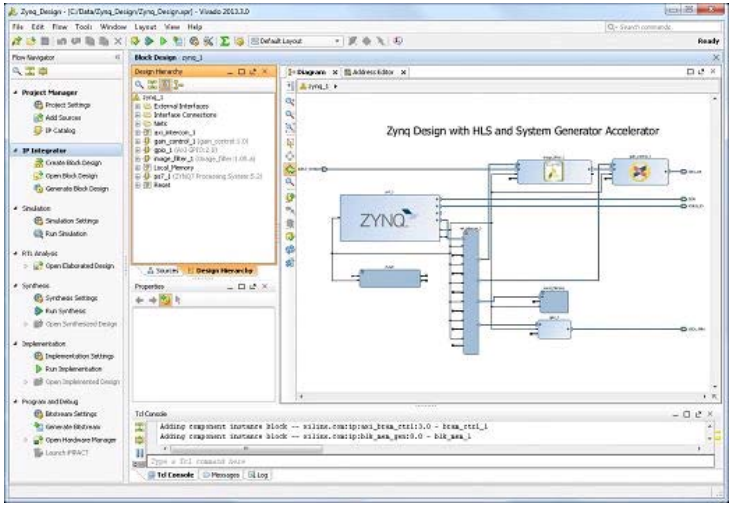
- Industrial automation software can be expressed as structured text (IEC 61131-3)
- Traditionally, however, PLC programmers use a graphical method of expressing programs; Ladder or Function blocks. This produces non-textual, sometimes binary and complex source “code”
- Special versioning software is used to manage the required versioning of non-textual industrial automation code
- However, many of the benefits of the Continuous Integration workflow are lost

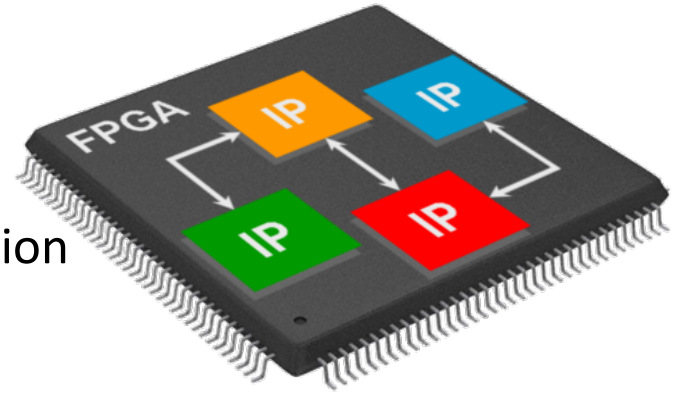


```
    END_IF;
Setpoint_IN_STAGE_1_FAILED:
    (* During 'STAGE_1_FAILED': '<S1>:119' *)
    IF (stage3_sensor <= 0) OR (stage2_sensor <= 0) THEN
        (* Transition: '(<S1>:150' *)
        (* Transition: '(<S1>:152' *)
        IF stage2_sensor > 0 THEN
            (* Transition: '<S1>:155' *)
            is_c2_Setpoint := Setpoint_IN_STAGES_1_3_FAILED;
            (* Entry 'STAGES_1_3_FAILED': '<S1>:120' *)
            rtb_stage1_setpoint := L0;
            rtb_stage2_setpoint := L0 - overall_target;
            distributed_target := rtb_stage2_setpoint;
        ELSE
            (* Transition: '<S1>:154' *)
            IF stage3_sensor > 0 THEN
                (* Transition: '<S1>:159' *)
                is_c2_Setpoint := Setpoint_IN_STAGES_1_2_FAILED;
                (* Entry 'STAGES_1_2_FAILED': '<S1>:121' *)
                rtb_stage1_setpoint := L0;
                rtb_stage2_setpoint := L0;
                distributed_target := L0 - overall_target;
            ELSE
                guard_0 := TRUE;
            END_IF;
        END_IF;
    ELSE
        guard_0 := TRUE;
    END_IF;
```

Configurable hardware

- FPGA firmware is to be expressed as VHDL text
- Some parts of the manufacturers development environment do not allow or promote full textual representation.
- Programmers instead use a graphical method of expressing functionality. This produces non-textual, sometimes binary and complex source “code”
- The current solution is to manage non-textual FPGA firmware code and configuration as “blobs” in a version management system
- However, many of the benefits of the Continuous Integration workflow are lost





Summary

- Software is an important part of the ESS facility
- Many separate software products are aggregated into systems critical for the operation of ESS
- We use modern software methodologies - continuous integration - to develop and deploy software systems at ESS
  - This is how we “commission” software at ESS
- We have a consistent development environment EEE for the distributed efforts in the project
- The ambition is to deal with “everything” as code including configuration information, industrial automation software and firmware

Thank you!