

# Big CI from scratch

*Or how I stopped worrying and started to love the automatic test*

March 26, 2021

# Disclaimer:

- ▶ Layman's experience from the trenches
- ▶ AKA: *Dauids opinion considered harmful!*

# Disclaimer:

- ▶ Layman's experience from the trenches
- ▶ AKA: *Dauids opinion considered harmful!*

## Disclaimer:

- ▶ Layman's experience from the trenches
- ▶ AKA: *Dauids opinion considered harmful!*

# Ericsson RBS



NEXXER

# Ericsson RBS

- ▶ Thousands of developers
- ▶ All developing for Radio Base Station
  - ▶ Different sub-organizations, different responsibilities
  - ▶ Nexer, one sub area  $\approx$  couple hundred developers
- ▶ Gerrit / Git / Jenkins / Jira / (Eiffel)
- ▶ + in-house tools

# Ericsson RBS

- ▶ Thousands of developers
- ▶ All developing for Radio Base Station
  - ▶ Different sub-organizations, different responsibilities
  - ▶ Nexer, one sub area  $\approx$  couple hundred developers
- ▶ Gerrit / Git / Jenkins / Jira / (Eiffel)
- ▶ + in-house tools

# Ericsson RBS

- ▶ Thousands of developers
- ▶ All developing for Radio Base Station
  - ▶ Different sub-organizations, different responsibilities
  - ▶ Nexer, one sub area  $\approx$  couple hundred developers
- ▶ Gerrit / Git / Jenkins / Jira / (Eiffel)
- ▶ + in-house tools



# Ericsson RBS

- ▶ Thousands of developers
- ▶ All developing for Radio Base Station
  - ▶ Different sub-organizations, different responsibilities
  - ▶ Nexer, one sub area  $\approx$  couple hundred developers
- ▶ Gerrit / Git / Jenkins / Jira / (Eiffel)
- ▶ + in-house tools

# Ericsson RBS

- ▶ Thousands of developers
- ▶ All developing for Radio Base Station
  - ▶ Different sub-organizations, different responsibilities
  - ▶ Nexer, one sub area  $\approx$  couple hundred developers
- ▶ Gerrit / Git / Jenkins / Jira / (Eiffel)
- ▶ + in-house tools

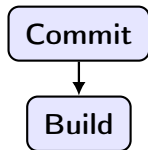
# Ericsson RBS

- ▶ Thousands of developers
- ▶ All developing for Radio Base Station
  - ▶ Different sub-organizations, different responsibilities
  - ▶ Nexer, one sub area  $\approx$  couple hundred developers
- ▶ Gerrit / Git / Jenkins / Jira / (Eiffel)
- ▶ + in-house tools

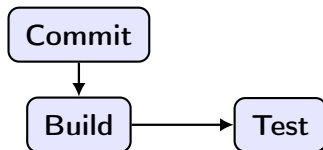
# Basic CI

Commit

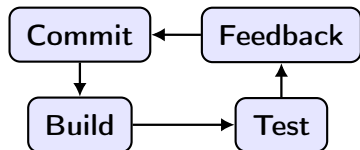
# Basic CI



# Basic CI



# Basic CI



# Big CI Problems

- ▶ Test scope
  - ▶ Can we run all tests?
  - ▶ Where should tests run?
  - ▶ Are all tests passing?
- ▶ Tracking
  - ▶ Where is my commit?
  - ▶ Is my commit ok?
- ▶ Intermittency
  - ▶ Lots of tests + intermittent tests  $\equiv$  no flow
- ▶ Lead time
  - ▶ Feedback loop
- ▶ Many developers
  - ▶  $\rightarrow$  Many Bottlenecks
  - ▶ Dependencies (expected and unexpected!)



# Big CI Problems

- ▶ Test scope
  - ▶ Can we run all tests?
  - ▶ Where should tests run?
  - ▶ Are all tests passing?
- ▶ Tracking
  - ▶ Where is my commit?
  - ▶ Is my commit ok?
- ▶ Intermittency
  - ▶ Lots of tests + intermittent tests  $\equiv$  no flow
- ▶ Lead time
  - ▶ Feedback loop
- ▶ Many developers
  - ▶  $\rightarrow$  Many Bottlenecks
  - ▶ Dependencies (expected and unexpected!)

# Big CI Problems

- ▶ Test scope
  - ▶ Can we run all tests?
  - ▶ Where should tests run?
  - ▶ Are all tests passing?
- ▶ Tracking
  - ▶ Where is my commit?
  - ▶ Is my commit ok?
- ▶ Intermittency
  - ▶ Lots of tests + intermittent tests  $\equiv$  no flow
- ▶ Lead time
  - ▶ Feedback loop
- ▶ Many developers
  - ▶  $\rightarrow$  Many Bottlenecks
  - ▶ Dependencies (expected and unexpected!)

# Big CI Problems

- ▶ Test scope
  - ▶ Can we run all tests?
  - ▶ Where should tests run?
  - ▶ Are all tests passing?
- ▶ Tracking
  - ▶ Where is my commit?
  - ▶ Is my commit ok?
- ▶ Intermittency
  - ▶ Lots of tests + intermittent tests  $\equiv$  no flow
- ▶ Lead time
  - ▶ Feedback loop
- ▶ Many developers
  - ▶  $\rightarrow$  Many Bottlenecks
  - ▶ Dependencies (expected and unexpected!)

# Big CI Problems

- ▶ Test scope
  - ▶ Can we run all tests?
  - ▶ Where should tests run?
  - ▶ Are all tests passing?
- ▶ Tracking
  - ▶ Where is my commit?
  - ▶ Is my commit ok?
- ▶ Intermittency
  - ▶ Lots of tests + intermittent tests  $\equiv$  no flow
- ▶ Lead time
  - ▶ Feedback loop
- ▶ Many developers
  - ▶  $\rightarrow$  Many Bottlenecks
  - ▶ Dependencies (expected and unexpected!)

# Big CI Problems

- ▶ Test scope
  - ▶ Can we run all tests?
  - ▶ Where should tests run?
  - ▶ Are all tests passing?
- ▶ Tracking
  - ▶ Where is my commit?
  - ▶ Is my commit ok?
- ▶ Intermittency
  - ▶ Lots of tests + intermittent tests  $\equiv$  no flow
- ▶ Lead time
  - ▶ Feedback loop
- ▶ Many developers
  - ▶  $\rightarrow$  Many Bottlenecks
  - ▶ Dependencies (expected and unexpected!)

# Big CI Problems

- ▶ Test scope
  - ▶ Can we run all tests?
  - ▶ Where should tests run?
  - ▶ Are all tests passing?
- ▶ Tracking
  - ▶ Where is my commit?
  - ▶ Is my commit ok?
- ▶ Intermittency
  - ▶ Lots of tests + intermittent tests  $\equiv$  no flow
- ▶ Lead time
  - ▶ Feedback loop
- ▶ Many developers
  - ▶  $\rightarrow$  Many Bottlenecks
  - ▶ Dependencies (expected and unexpected!)

# Big CI Problems

- ▶ Test scope
  - ▶ Can we run all tests?
  - ▶ Where should tests run?
  - ▶ Are all tests passing?
- ▶ Tracking
  - ▶ Where is my commit?
  - ▶ Is my commit ok?
- ▶ Intermittency
  - ▶ Lots of tests + intermittent tests  $\equiv$  no flow
- ▶ Lead time
  - ▶ Feedback loop
- ▶ Many developers
  - ▶  $\rightarrow$  Many Bottlenecks
  - ▶ Dependencies (expected and unexpected!)

# Big CI Problems

- ▶ Test scope
  - ▶ Can we run all tests?
  - ▶ Where should tests run?
  - ▶ Are all tests passing?
- ▶ Tracking
  - ▶ Where is my commit?
  - ▶ Is my commit ok?
- ▶ Intermittency
  - ▶ Lots of tests + intermittent tests  $\equiv$  no flow
- ▶ Lead time
  - ▶ Feedback loop
- ▶ Many developers
  - ▶  $\rightarrow$  Many Bottlenecks
  - ▶ Dependencies (expected and unexpected!)



# Big CI Problems

- ▶ Test scope
  - ▶ Can we run all tests?
  - ▶ Where should tests run?
  - ▶ Are all tests passing?
- ▶ Tracking
  - ▶ Where is my commit?
  - ▶ Is my commit ok?
- ▶ Intermittency
  - ▶ Lots of tests + intermittent tests  $\equiv$  no flow
- ▶ Lead time
  - ▶ Feedback loop
- ▶ Many developers
  - ▶  $\rightarrow$  Many Bottlenecks
  - ▶ Dependencies (expected and unexpected!)

# Big CI Problems

- ▶ Test scope
  - ▶ Can we run all tests?
  - ▶ Where should tests run?
  - ▶ Are all tests passing?
- ▶ Tracking
  - ▶ Where is my commit?
  - ▶ Is my commit ok?
- ▶ Intermittency
  - ▶ Lots of tests + intermittent tests  $\equiv$  no flow
- ▶ Lead time
  - ▶ Feedback loop
- ▶ Many developers
  - ▶  $\rightarrow$  Many Bottlenecks
  - ▶ Dependencies (expected and unexpected!)

# Big CI Problems

- ▶ Test scope
  - ▶ Can we run all tests?
  - ▶ Where should tests run?
  - ▶ Are all tests passing?
- ▶ Tracking
  - ▶ Where is my commit?
  - ▶ Is my commit ok?
- ▶ Intermittency
  - ▶ Lots of tests + intermittent tests  $\equiv$  no flow
- ▶ Lead time
  - ▶ Feedback loop
- ▶ Many developers
  - ▶  $\rightarrow$  Many Bottlenecks
  - ▶ Dependencies (expected and unexpected!)

# Big CI Problems

- ▶ Test scope
  - ▶ Can we run all tests?
  - ▶ Where should tests run?
  - ▶ Are all tests passing?
- ▶ Tracking
  - ▶ Where is my commit?
  - ▶ Is my commit ok?
- ▶ Intermittency
  - ▶ Lots of tests + intermittent tests  $\equiv$  no flow
- ▶ Lead time
  - ▶ Feedback loop
- ▶ Many developers
  - ▶  $\rightarrow$  Many Bottlenecks
  - ▶ Dependencies (expected and unexpected!)

# Big CI Problems

- ▶ Test scope
  - ▶ Can we run all tests?
  - ▶ Where should tests run?
  - ▶ Are all tests passing?
- ▶ Tracking
  - ▶ Where is my commit?
  - ▶ Is my commit ok?
- ▶ Intermittency
  - ▶ Lots of tests + intermittent tests  $\equiv$  no flow
- ▶ Lead time
  - ▶ Feedback loop
- ▶ Many developers
  - ▶  $\rightarrow$  Many Bottlenecks
  - ▶ Dependencies (expected and unexpected!)

# Big CI Problems

- ▶ Test scope
  - ▶ Can we run all tests?
  - ▶ Where should tests run?
  - ▶ Are all tests passing?
- ▶ Tracking
  - ▶ Where is my commit?
  - ▶ Is my commit ok?
- ▶ Intermittency
  - ▶ Lots of tests + intermittent tests  $\equiv$  no flow
- ▶ Lead time
  - ▶ Feedback loop
- ▶ Many developers
  - ▶  $\rightarrow$  Many Bottlenecks
  - ▶ Dependencies (expected and unexpected!)

# CI from the CI Flow Plumber's point of view

- ▶ **Modularization**
- ▶ Logging
- ▶ Non-exhaustive list!
  - ▶ Speed
  - ▶ Stability
  - ▶ Reproducibility
  - ▶ Scalability
  - ▶ ...

# CI from the CI Flow Plumber's point of view

- ▶ Modularization
- ▶ Logging
- ▶ Non-exhaustive list!
  - ▶ Speed
  - ▶ Stability
  - ▶ Reproducibility
  - ▶ Scalability
  - ▶ ...



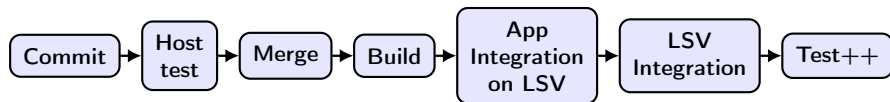
# CI from the CI Flow Plumber's point of view

- ▶ Modularization
- ▶ Logging
- ▶ Non-exhaustive list!
  - ▶ Speed
  - ▶ Stability
  - ▶ Reproducibility
  - ▶ Scalability
  - ▶ ...

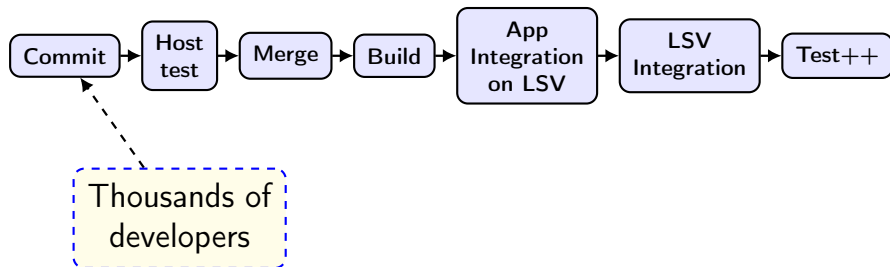
# CI from the CI Flow Plumber's point of view

- ▶ Modularization
- ▶ Logging
- ▶ Non-exhaustive list!
  - ▶ Speed
  - ▶ Stability
  - ▶ Reproducibility
  - ▶ Scalability
  - ▶ ...

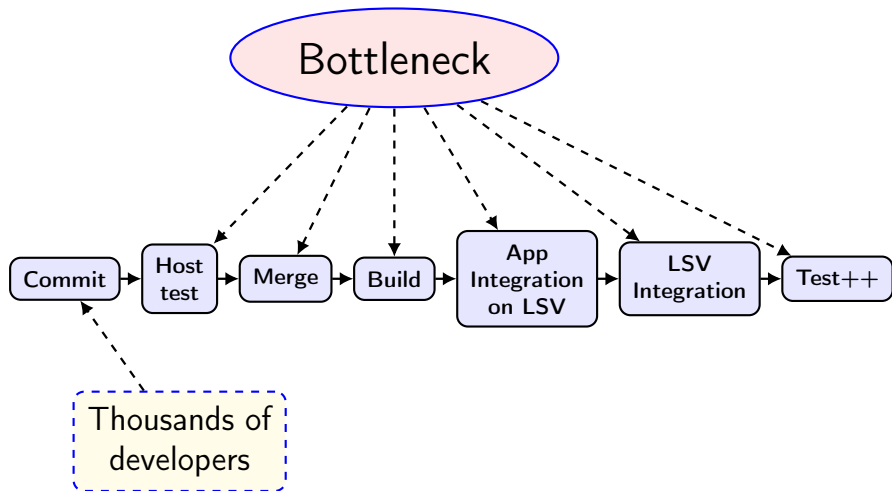
# Modularization



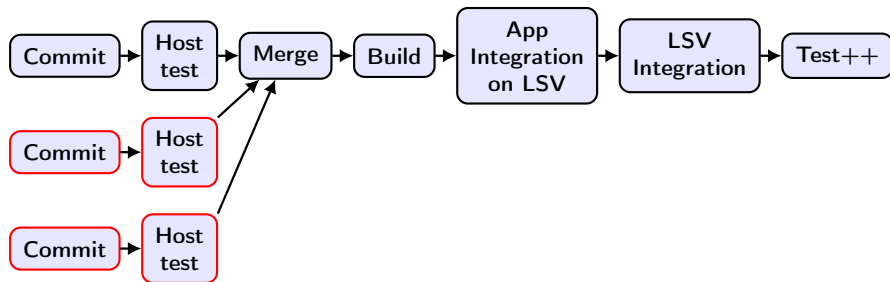
# Modularization



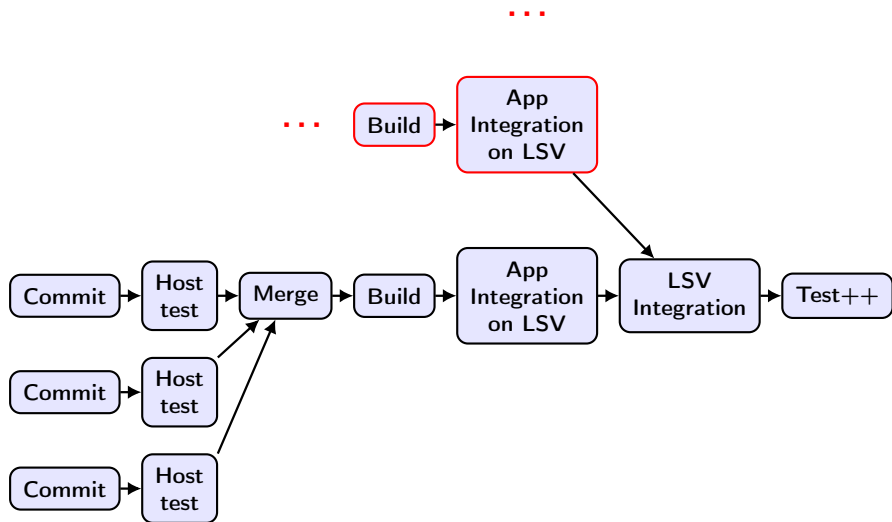
# Modularization



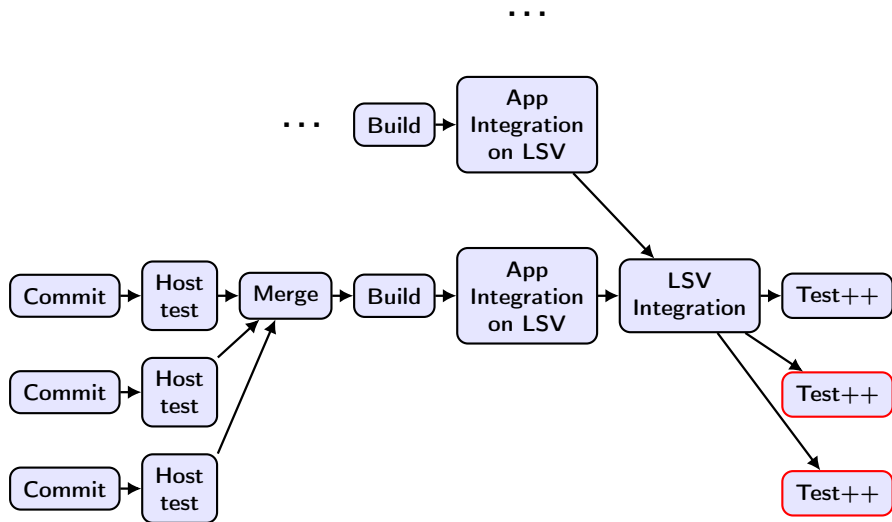
# Modularization



# Modularization

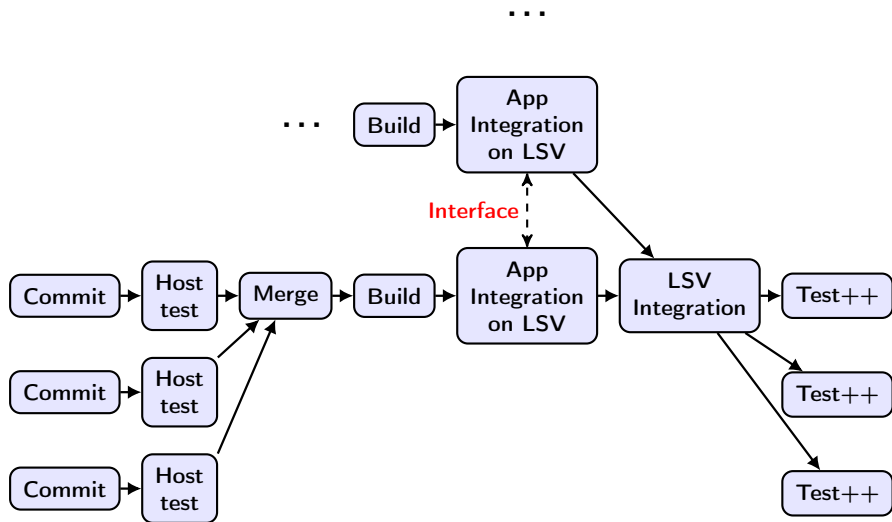


# Modularization

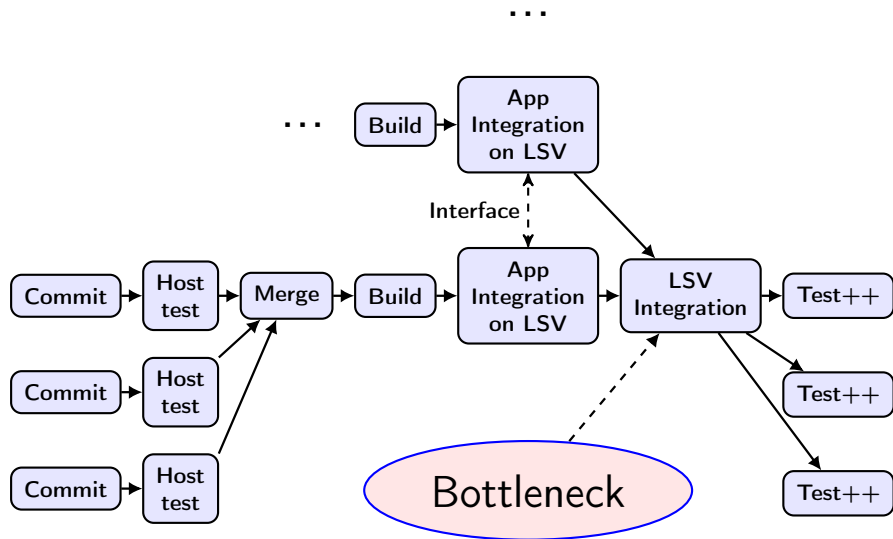




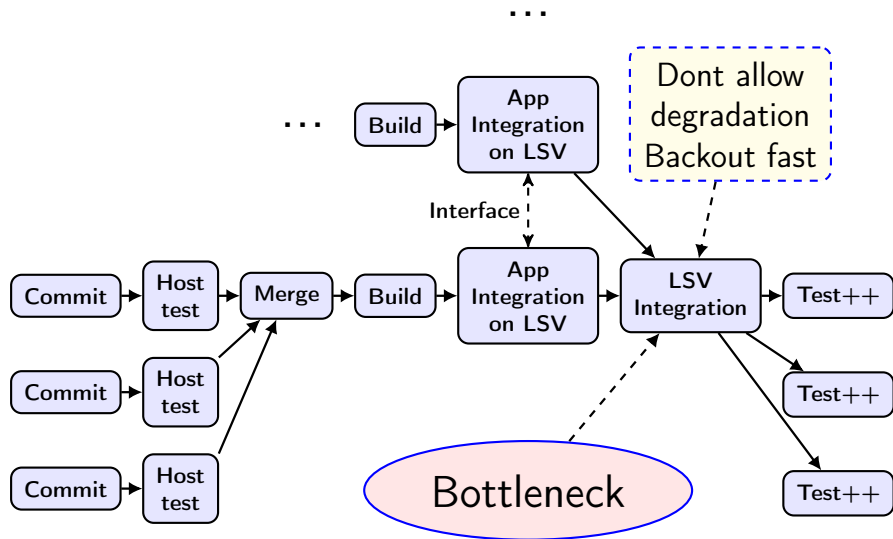
# Modularization



# Modularization



# Modularization



# Modularization thinking

- ▶ One developer/app should not stop flow for all
  - ▶ Bad quality → You don't get to play
  - ▶ Revert/recover first, fix later
- ▶ Needed:
  - ▶ Clean interfaces
  - ▶ Requirements
- ▶ (Enabler of Agile!)
  - ▶ More defined "sub" responsibilities, better backlogs
  - ▶ Sub-orgs solve similar problems → best solution wins!

# Modularization thinking

- ▶ One developer/app should not stop flow for all
  - ▶ Bad quality → You don't get to play
  - ▶ Revert/recover first, fix later
- ▶ Needed:
  - ▶ Clean interfaces
  - ▶ Requirements
- ▶ (Enabler of Agile!)
  - ▶ More defined "sub" responsibilities, better backlogs
  - ▶ Sub-orgs solve similar problems → best solution wins!

# Modularization thinking

- ▶ One developer/app should not stop flow for all
  - ▶ Bad quality → You don't get to play
  - ▶ Revert/recover first, fix later
- ▶ Needed:
  - ▶ Clean interfaces
  - ▶ Requirements
- ▶ (Enabler of Agile!)
  - ▶ More defined "sub" responsibilities, better backlogs
  - ▶ Sub-orgs solve similar problems → best solution wins!

# Modularization thinking

- ▶ One developer/app should not stop flow for all
  - ▶ Bad quality → You dont get to play
  - ▶ Revert/recover first, fix later
- ▶ Needed:
  - ▶ Clean interfaces
  - ▶ Requirements
- ▶ (Enabler of Agile!)
  - ▶ More defined "sub" responsibilities, better backlogs
  - ▶ Sub-orgs solve similar problems → best solution wins!

# Modularization thinking

- ▶ One developer/app should not stop flow for all
  - ▶ Bad quality → You don't get to play
  - ▶ Revert/recover first, fix later
- ▶ Needed:
  - ▶ Clean interfaces
  - ▶ Requirements
- ▶ (Enabler of Agile!)
  - ▶ More defined "sub" responsibilities, better backlogs
  - ▶ Sub-orgs solve similar problems → best solution wins!



# Modularization thinking

- ▶ One developer/app should not stop flow for all
  - ▶ Bad quality → You don't get to play
  - ▶ Revert/recover first, fix later
- ▶ Needed:
  - ▶ Clean interfaces
  - ▶ Requirements
- ▶ (Enabler of Agile!)
  - ▶ More defined "sub" responsibilities, better backlogs
  - ▶ Sub-orgs solve similar problems → best solution wins!

# Modularization thinking

- ▶ One developer/app should not stop flow for all
  - ▶ Bad quality → You don't get to play
  - ▶ Revert/recover first, fix later
- ▶ Needed:
  - ▶ Clean interfaces
  - ▶ Requirements
- ▶ (Enabler of Agile!)
  - ▶ More defined "sub" responsibilities, better backlogs
  - ▶ Sub-orgs solve similar problems → best solution wins!

# Modularization thinking

- ▶ One developer/app should not stop flow for all
  - ▶ Bad quality → You don't get to play
  - ▶ Revert/recover first, fix later
- ▶ Needed:
  - ▶ Clean interfaces
  - ▶ Requirements
- ▶ (Enabler of Agile!)
  - ▶ More defined "sub" responsibilities, better backlogs
  - ▶ Sub-orgs solve similar problems → best solution wins!

# Modularization thinking

- ▶ One developer/app should not stop flow for all
  - ▶ Bad quality → You don't get to play
  - ▶ Revert/recover first, fix later
- ▶ Needed:
  - ▶ Clean interfaces
  - ▶ Requirements
- ▶ (Enabler of Agile!)
  - ▶ More defined "sub" responsibilities, better backlogs
  - ▶ Sub-orgs solve similar problems → best solution wins!

# Modularization thinking

- ▶ One developer/app should not stop flow for all
  - ▶ Bad quality → You don't get to play
  - ▶ Revert/recover first, fix later
- ▶ Needed:
  - ▶ Clean interfaces
  - ▶ Requirements
- ▶ (Enabler of Agile!)
  - ▶ More defined "sub" responsibilities, better backlogs
  - ▶ Sub-orgs solve similar problems → best solution wins!

# Modularization bad things

- ▶ More spread out
  - ▶ Harder to cooperate
  - ▶ Multiple solutions to same problem (alignment)
- ▶ Permissions
  - ▶ *"Why should you have access to my code?"*
- ▶ "Box thinking"
  - ▶ "My box is perfect" → someone else's problem
  - ▶ Remember: All working for same goal

# Modularization bad things

- ▶ More spread out
  - ▶ Harder to cooperate
  - ▶ Multiple solutions to same problem (alignment)
- ▶ Permissions
  - ▶ *"Why should you have access to my code?"*
- ▶ "Box thinking"
  - ▶ "My box is perfect" → someone else's problem
  - ▶ Remember: All working for same goal

# Modularization bad things

- ▶ More spread out
  - ▶ Harder to cooperate
  - ▶ Multiple solutions to same problem (alignment)
- ▶ Permissions
  - ▶ *"Why should you have access to my code?"*
- ▶ "Box thinking"
  - ▶ "My box is perfect" → someone else's problem
  - ▶ Remember: All working for same goal



# Modularization bad things

- ▶ More spread out
  - ▶ Harder to cooperate
  - ▶ Multiple solutions to same problem (alignment)
- ▶ Permissions
  - ▶ *"Why should you have access to my code?"*
- ▶ "Box thinking"
  - ▶ "My box is perfect" → someone else's problem
  - ▶ Remember: All working for same goal

# Modularization bad things

- ▶ More spread out
  - ▶ Harder to cooperate
  - ▶ Multiple solutions to same problem (alignment)
- ▶ Permissions
  - ▶ "*Why should you have access to my code?*"
- ▶ "Box thinking"
  - ▶ "My box is perfect" → someone else's problem
  - ▶ Remember: All working for same goal

# Modularization bad things

- ▶ More spread out
  - ▶ Harder to cooperate
  - ▶ Multiple solutions to same problem (alignment)
- ▶ Permissions
  - ▶ "*Why should you have access to my code?*"
- ▶ "Box thinking"
  - ▶ "My box is perfect" → someone else's problem
  - ▶ Remember: All working for same goal

# Modularization bad things

- ▶ More spread out
  - ▶ Harder to cooperate
  - ▶ Multiple solutions to same problem (alignment)
- ▶ Permissions
  - ▶ *"Why should you have access to my code?"*
- ▶ "Box thinking"
  - ▶ "My box is perfect" → someone else's problem
  - ▶ Remember: All working for same goal

# Modularization bad things



# Modularization bad things

- ▶ More spread out
  - ▶ Harder to cooperate
  - ▶ Multiple solutions to same problem (alignment)
- ▶ Permissions
  - ▶ "*Why should you have access to my code?*"
- ▶ "Box thinking"
  - ▶ "My box is perfect" → someone else's problem
  - ▶ Remember: All working for same goal

# Modularization bad things

- ▶ More spread out
  - ▶ Harder to cooperate
  - ▶ Multiple solutions to same problem (alignment)
- ▶ Permissions
  - ▶ *"Why should you have access to my code?"*
- ▶ "Box thinking"
  - ▶ "My box is perfect" → someone else's problem
  - ▶ Remember: **All working for same goal**

# Way towards Big CI, early actions

## 1. Parallelism

- ▶ Enables running many tests

## 2. Build avoidance / caching

- ▶ Don't rebuild source/objects that have not changed
- ▶ Cache objects/build dependencies between consecutive runs

## 3. Smart testing

- ▶ Many tests → running all cripples CI
- ▶ Only run tests that are related to change

## 4. Invest in Application and CI architecture

- ▶ Design for testability
  - ▶ Divide application into sub responsibilities (modularization)
  - ▶ Communicate with backwards compatible interfaces
  - ▶ Separation of concerns!



# Way towards Big CI, early actions

## 1. Parallelism

- ▶ Enables running many tests

## 2. Build avoidance / caching

- ▶ Don't rebuild source/objects that have not changed
- ▶ Cache objects/build dependencies between consecutive runs

## 3. Smart testing

- ▶ Many tests → running all cripples CI
- ▶ Only run tests that are related to change

## 4. Invest in Application and CI architecture

- ▶ Design for testability
  - ▶ Divide application into sub responsibilities (modularization)
  - ▶ Communicate with backwards compatible interfaces
  - ▶ Separation of concerns!

# Way towards Big CI, early actions

## 1. Parallelism

- ▶ Enables running many tests

## 2. Build avoidance / caching

- ▶ Don't rebuild source/objects that have not changed
- ▶ Cache objects/build dependencies between consecutive runs

## 3. Smart testing

- ▶ Many tests → running all cripples CI
- ▶ Only run tests that are related to change

## 4. Invest in Application and CI architecture

- ▶ Design for testability
  - ▶ Divide application into sub responsibilities (modularization)
  - ▶ Communicate with backwards compatible interfaces
  - ▶ Separation of concerns!

# Way towards Big CI, early actions

## 1. Parallelism

- ▶ Enables running many tests

## 2. Build avoidance / caching

- ▶ Don't rebuild source/objects that have not changed
- ▶ Cache objects/build dependencies between consecutive runs

## 3. Smart testing

- ▶ Many tests → running all cripples CI
- ▶ Only run tests that are related to change

## 4. Invest in Application and CI architecture

- ▶ Design for testability
  - ▶ Divide application into sub responsibilities (modularization)
  - ▶ Communicate with backwards compatible interfaces
  - ▶ Separation of concerns!

# Way towards Big CI, early actions

## 1. Parallelism

- ▶ Enables running many tests

## 2. Build avoidance / caching

- ▶ Don't rebuild source/objects that have not changed
- ▶ Cache objects/build dependencies between consecutive runs

## 3. Smart testing

- ▶ Many tests → running all cripples CI
- ▶ Only run tests that are related to change

## 4. Invest in Application and CI architecture

- ▶ Design for testability
  - ▶ Divide application into sub responsibilities (modularization)
  - ▶ Communicate with backwards compatible interfaces
  - ▶ Separation of concerns!

# Way towards Big CI, early actions

## 1. Parallelism

- ▶ Enables running many tests

## 2. Build avoidance / caching

- ▶ Don't rebuild source/objects that have not changed
- ▶ Cache objects/build dependencies between consecutive runs

## 3. Smart testing

- ▶ Many tests → running all cripples CI
- ▶ Only run tests that are related to change

## 4. Invest in Application and CI architecture

- ▶ Design for testability
  - ▶ Divide application into sub responsibilities (modularization)
  - ▶ Communicate with backwards compatible interfaces
  - ▶ Separation of concerns!

# Way towards Big CI, early actions

## 1. Parallelism

- ▶ Enables running many tests

## 2. Build avoidance / caching

- ▶ Don't rebuild source/objects that have not changed
- ▶ Cache objects/build dependencies between consecutive runs

## 3. Smart testing

- ▶ Many tests → running all cripples CI
- ▶ Only run tests that are related to change

## 4. Invest in Application and CI architecture

- ▶ Design for testability
  - ▶ Divide application into sub responsibilities (modularization)
  - ▶ Communicate with backwards compatible interfaces
  - ▶ Separation of concerns!

# Way towards Big CI, early actions

1. Parallelism
  - ▶ Enables running many tests
2. Build avoidance / caching
  - ▶ Don't rebuild source/objects that have not changed
  - ▶ Cache objects/build dependencies between consecutive runs
3. Smart testing
  - ▶ Many tests → running all cripples CI
  - ▶ Only run tests that are related to change
4. Invest in Application and CI architecture
  - ▶ Design for testability
    - ▶ Divide application into sub responsibilities (modularization)
    - ▶ Communicate with backwards compatible interfaces
    - ▶ Separation of concerns!

# Way towards Big CI, early actions

1. Parallelism
  - ▶ Enables running many tests
2. Build avoidance / caching
  - ▶ Don't rebuild source/objects that have not changed
  - ▶ Cache objects/build dependencies between consecutive runs
3. Smart testing
  - ▶ Many tests → running all cripples CI
  - ▶ Only run tests that are related to change
4. Invest in Application **and** CI architecture
  - ▶ Design for testability
    - ▶ Divide application into sub responsibilities (modularization)
    - ▶ Communicate with backwards compatible interfaces
    - ▶ Separation of concerns!



# Way towards Big CI, early actions

1. Parallelism
  - ▶ Enables running many tests
2. Build avoidance / caching
  - ▶ Don't rebuild source/objects that have not changed
  - ▶ Cache objects/build dependencies between consecutive runs
3. Smart testing
  - ▶ Many tests → running all cripples CI
  - ▶ Only run tests that are related to change
4. Invest in Application and CI architecture
  - ▶ **Design for testability**
    - ▶ Divide application into sub responsibilities (modularization)
    - ▶ Communicate with backwards compatible interfaces
    - ▶ Separation of concerns!

# Way towards Big CI, early actions

1. Parallelism
  - ▶ Enables running many tests
2. Build avoidance / caching
  - ▶ Don't rebuild source/objects that have not changed
  - ▶ Cache objects/build dependencies between consecutive runs
3. Smart testing
  - ▶ Many tests → running all cripples CI
  - ▶ Only run tests that are related to change
4. Invest in Application and CI architecture
  - ▶ Design for testability
    - ▶ Divide application into sub responsibilities (modularization)
    - ▶ Communicate with backwards compatible interfaces
    - ▶ Separation of concerns!

# Way towards Big CI, early actions

1. Parallelism
  - ▶ Enables running many tests
2. Build avoidance / caching
  - ▶ Don't rebuild source/objects that have not changed
  - ▶ Cache objects/build dependencies between consecutive runs
3. Smart testing
  - ▶ Many tests → running all cripples CI
  - ▶ Only run tests that are related to change
4. Invest in Application and CI architecture
  - ▶ Design for testability
    - ▶ Divide application into sub responsibilities (modularization)
    - ▶ Communicate with backwards compatible interfaces
    - ▶ Separation of concerns!

# Way towards Big CI, early actions

1. Parallelism
  - ▶ Enables running many tests
2. Build avoidance / caching
  - ▶ Don't rebuild source/objects that have not changed
  - ▶ Cache objects/build dependencies between consecutive runs
3. Smart testing
  - ▶ Many tests → running all cripples CI
  - ▶ Only run tests that are related to change
4. Invest in Application and CI architecture
  - ▶ Design for testability
    - ▶ Divide application into sub responsibilities (modularization)
    - ▶ Communicate with backwards compatible interfaces
    - ▶ Separation of concerns!

# Way towards Big CI, early actions

## 1. Parallelism

- ▶ Enables running many tests

## 2. Build avoidance / caching

- ▶ Don't rebuild source/objects that have not changed
- ▶ Cache objects/build dependencies between consecutive runs

## 3. Smart testing

- ▶ Many tests → running all cripples CI
- ▶ Only run tests that are related to change

## 4. Invest in Application and CI architecture

- ▶ Design for testability
  - ▶ Divide application into sub responsibilities (modularization)
  - ▶ Communicate with backwards compatible interfaces
  - ▶ Separation of concerns!

# Running all the tests

```
$ cd project-x
$ . ci/setup.sh
$ time apps/app00/test/test.sh
## Running tests for /home/solarus/projects/project-x/apps/app00
# Doing complicated arithmetic (aka sleeping) for 8 seconds ...
# Done!
real 0m8.014s
```

```
$ time find -name test.sh -exec {} \;
## Running tests for /home/solarus/projects/project-x/apps/app04
# Doing complicated arithmetic (aka sleeping) for 0 seconds ...
# Done!
...
## Running tests for /home/solarus/projects/project-x/apps/app03
# Doing complicated arithmetic (aka sleeping) for 28 seconds ...
# Done!
real 11m13.586s
```

# Running all the tests

```
$ cd project-x
$ . ci/setup.sh
$ time apps/app00/test/test.sh
## Running tests for /home/solarus/projects/project-x/apps/app00
# Doing complicated arithmetic (aka sleeping) for 8 seconds ...
# Done!
real 0m8.014s
```

```
$ time find -name test.sh -exec {} \;
## Running tests for /home/solarus/projects/project-x/apps/app04
# Doing complicated arithmetic (aka sleeping) for 0 seconds ...
# Done!
...
## Running tests for /home/solarus/projects/project-x/apps/app03
# Doing complicated arithmetic (aka sleeping) for 28 seconds ...
# Done!
real 11m13.586s
```

# Running all tests

- ▶ In this case 50 suites
  - ▶ Around 15 seconds to finish → on average 12.5 minutes running sequentially
- ▶ Example from one repository:
  - ▶ 1 929 test suites
  - ▶ (1 035 437 lines of test code)
- ▶ Around 15 seconds to finish → about 482 minutes of sequential run time
  - ▶ I.e. a work day...



# Running all tests

- ▶ In this case 50 suites
  - ▶ Around 15 seconds to finish → on average 12.5 minutes running sequentially
- ▶ Example from one repository:
  - ▶ 1 929 test suites
  - ▶ (1 035 437 lines of test code)
- ▶ Around 15 seconds to finish → about 482 minutes of sequential run time
  - ▶ I.e. a work day...

# Running all tests

- ▶ In this case 50 suites
  - ▶ Around 15 seconds to finish → on average 12.5 minutes running sequentially
- ▶ Example from one repository:
  - ▶ 1 929 test suites
  - ▶ (1 035 437 lines of test code)
- ▶ Around 15 seconds to finish → about 482 minutes of sequential run time
  - ▶ I.e. a work day...

# Running all tests

- ▶ In this case 50 suites
  - ▶ Around 15 seconds to finish → on average 12.5 minutes running sequentially
- ▶ Example from one repository:
  - ▶ 1 929 test suites
  - ▶ (1 035 437 lines of **test** code)
- ▶ Around 15 seconds to finish → about 482 minutes of sequential run time
  - ▶ I.e. a work day...

# Running all tests

- ▶ In this case 50 suites
  - ▶ Around 15 seconds to finish → on average 12.5 minutes running sequentially
- ▶ Example from **one** repository:
  - ▶ 1 929 test suites
  - ▶ (1 035 437 lines of **test** code)
- ▶ Around 15 seconds to finish → about 482 minutes of sequential run time
  - ▶ I.e. a work day...

# Running all tests

- ▶ In this case 50 suites
  - ▶ Around 15 seconds to finish → on average 12.5 minutes running sequentially
- ▶ Example from one repository:
  - ▶ 1 929 test suites
  - ▶ (1 035 437 lines of **test** code)
- ▶ Around 15 seconds to finish → about 482 minutes of sequential run time
  - ▶ I.e. a work day...

# Running all tests

- ▶ In this case 50 suites
  - ▶ Around 15 seconds to finish → on average 12.5 minutes running sequentially
- ▶ Example from one repository:
  - ▶ 1 929 test suites
  - ▶ (1 035 437 lines of **test** code)
- ▶ Around 15 seconds to finish → about 482 minutes of sequential run time
  - ▶ I.e. a work day...

Questions?



# Logging

- ▶ Remember Big CI Problems:
  - ▶ Many tests+developers+apps/Tracking/Intermittency...
- ▶ Test failed in App Integration →
  - ▶ Test failed before? (same way!)
    - ▶ In same App/other apps?
    - ▶ On certain configurations?
  - ▶ Intermittent?
    - ▶ More intermittent today than last week?



# Logging

- ▶ Remember Big CI Problems:
  - ▶ Many tests+developers+apps/Tracking/Intermittency...
- ▶ Test failed in App Integration →
  - ▶ Test failed before? (same way!)
    - ▶ In same App/other apps?
    - ▶ On certain configurations?
  - ▶ Intermittent?
    - ▶ More intermittent today than last week?

# Logging

- ▶ Remember Big CI Problems:
  - ▶ Many tests+developers+apps/Tracking/Intermittency...
- ▶ Test failed in App Integration →
  - ▶ Test failed before? (same way!)
    - ▶ In same App/other apps?
    - ▶ On certain configurations?
  - ▶ Intermittent?
    - ▶ More intermittent today than last week?

# Logging

- ▶ Remember Big CI Problems:
  - ▶ Many tests+developers+apps/Tracking/Intermittency...
- ▶ Test failed in App Integration →
  - ▶ Test failed before? (same way!)
    - ▶ In same App/other apps?
    - ▶ On certain configurations?
  - ▶ Intermittent?
    - ▶ More intermittent today than last week?

# Logging

- ▶ Remember Big CI Problems:
  - ▶ Many tests+developers+apps/Tracking/Intermittency...
- ▶ Test failed in App Integration →
  - ▶ Test failed before? (same way!)
    - ▶ In same App/other apps?
    - ▶ On certain configurations?
  - ▶ Intermittent?
    - ▶ More intermittent today than last week?

# Logging

- ▶ Remember Big CI Problems:
  - ▶ Many tests+developers+apps/Tracking/Intermittency...
- ▶ Test failed in App Integration →
  - ▶ Test failed before? (same way!)
    - ▶ In same App/other apps?
    - ▶ On certain configurations?
  - ▶ Intermittent?
    - ▶ More intermittent today than last week?

# Logging

- ▶ Remember Big CI Problems:
  - ▶ Many tests+developers+apps/Tracking/Intermittency...
- ▶ Test failed in App Integration →
  - ▶ Test failed before? (same way!)
    - ▶ In same App/other apps?
    - ▶ On certain configurations?
  - ▶ Intermittent?
    - ▶ More intermittent today than last week?

# Logging

- ▶ Remember Big CI Problems:
  - ▶ Many tests+developers+apps/Tracking/Intermittency...
- ▶ Test failed in App Integration →
  - ▶ Test failed before? (same way!)
    - ▶ In same App/other apps?
    - ▶ On certain configurations?
  - ▶ Intermittent?
    - ▶ More intermittent today than last week?

# Logging cont

- ▶ Without data, we are blind to degradations
- ▶ Solution: automatic result tracking!
  - ▶ Test failure messages, configurations, target log analysis



# Logging cont

- ▶ Without data, we are blind to degradations
- ▶ **Solution: automatic result tracking!**
  - ▶ Test failure messages, configurations, target log analysis

## Logging cont

- ▶ Without data, we are blind to degradations
- ▶ Solution: automatic result tracking!
  - ▶ Test failure messages, configurations, target log analysis