Continuous Integration to compile and test Navit

Patrick Höhn
Navit Project
hoehnp@gmx.de

Outline

- Introduction
- Build-Infrastructure
- Continuous Integration
- Device Farm
- Appium
- Conclusion and Future Work

Outline

- **Introduction**
- Build-Infrastructure
- Continuous Integration
- Device Farm
- Appium
- Conclusion and Future Work

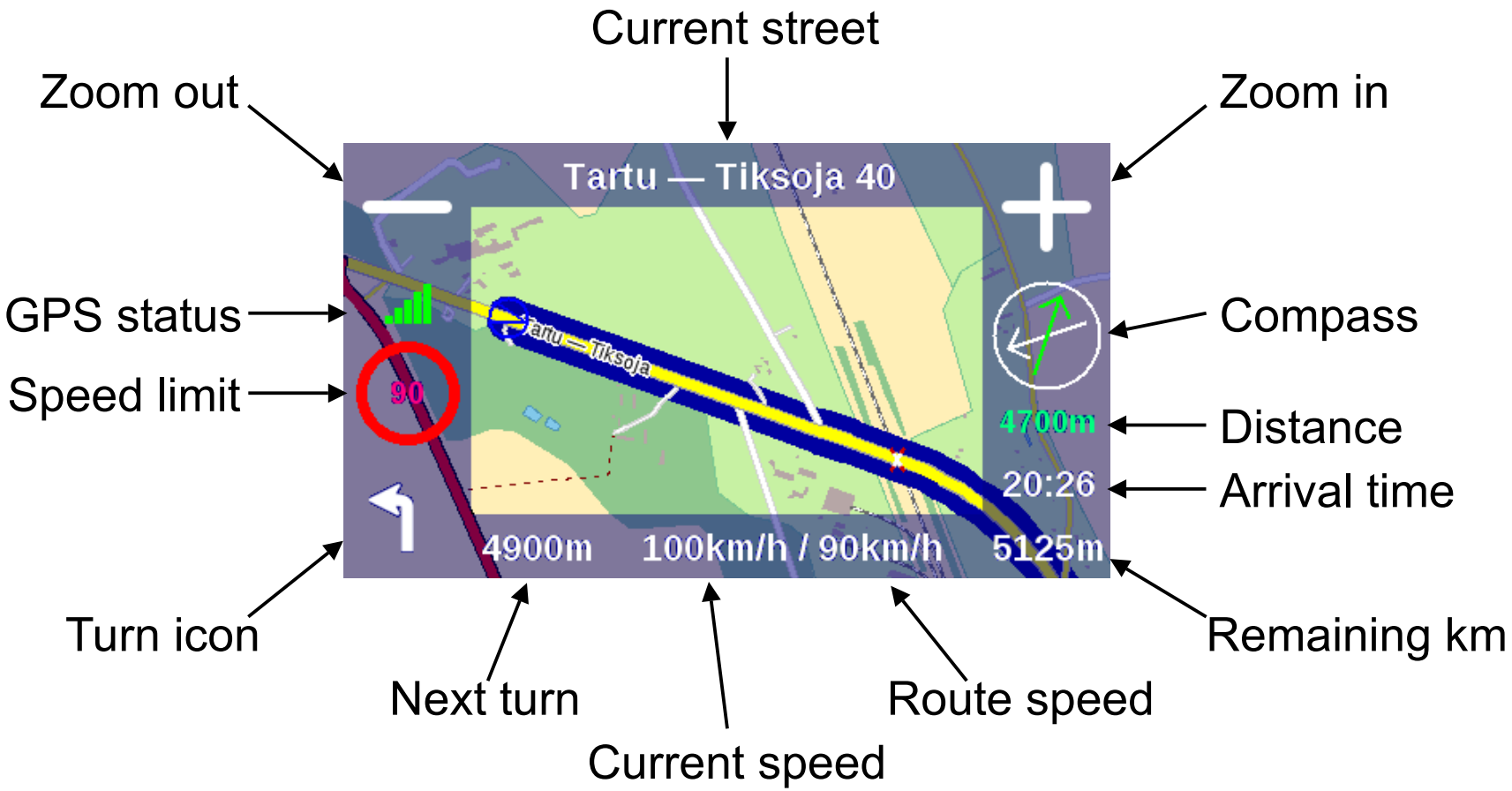
What is Navit?

- Multi-platform offline routing software
- Open source since 10+ years
- Usage of free OpenStreetMap data
- Advantages for end users:
  - Low system requirements (tested on 64MB RAM, 400MHz ARMv5)
  - Many supported platforms and devices
  - Highly customizable:
    - map layout
    - on-screen display
    - routing profiles
  - Several text-to-speech backends
  - Translations in more than 50 languages

Supported Platforms

- Laptop
  - Linux
  - Windows
  - MacOS
- Handheld
  - Android
  - Sailfish OS
  - WinCE
- Embedded
  - TomTom
  - Raspberry Pi

On the Road


Current street
Zoom out
Zoom in
Tartu — Tiksoja 40
GPS status
Compass
Speed limit
90
4700m
Distance
20:26
Arrival time
4900m
100km/h / 90km/h
5125m
Turn icon
Remaining km
Next turn
Route speed
Current speed

Outline

- Introduction
- **Build-Infrastructure**
- Continuous Integration
- Device Farm
- Appium
- Conclusion and Future Work

Build infrastructure

Challenges for Open Source projects

- Most contributors are interested in the project, not its infrastructure
- You need resources to host the services
- You often need money to pay for resources

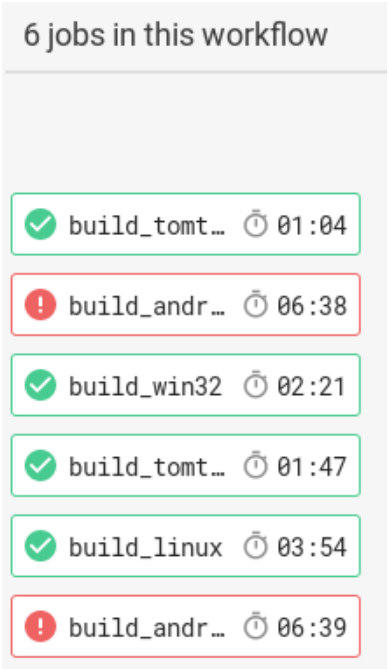
Build infrastructure

There is hope!
Several CI systems offer a free tier

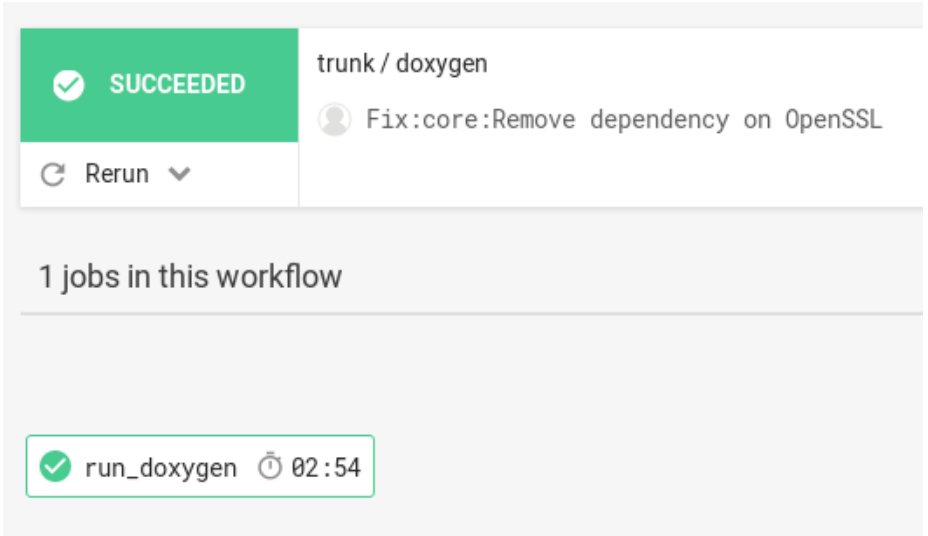
- compilation tests. (code base broken?)
- Run platform specific tests. (build for platform broken?)
- Run static code analysis. (no new issues?)
- Chain all the things together

CI – Build test pipeline

Parallel Jobs:



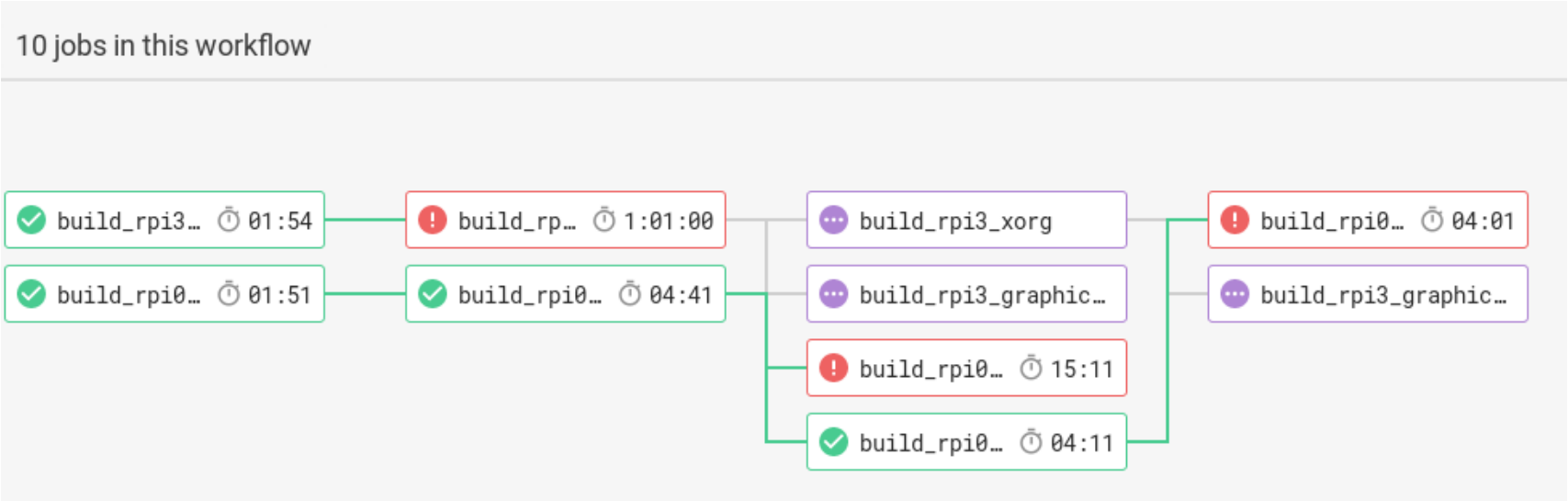
Branch-specific workflows, e.g. only rebuild our doxygen documentation from `trunk`:



CI – Build tests pipeline

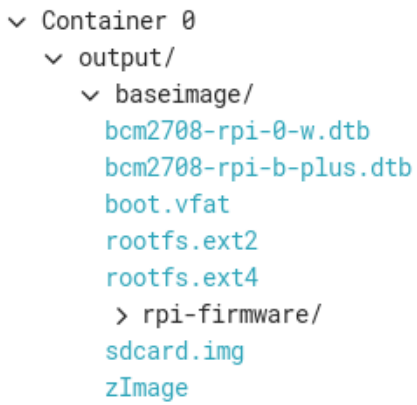
Advanced features workflow

- e.g. fan out/fan in.
- Useful for a job depending on another jobs earlier in the chain
- Location of build failure easily noticeable



Build artifacts

- Pipeline used to generate artifacts
  - e.g. generation of an image to installation of navit on Raspberry Pi



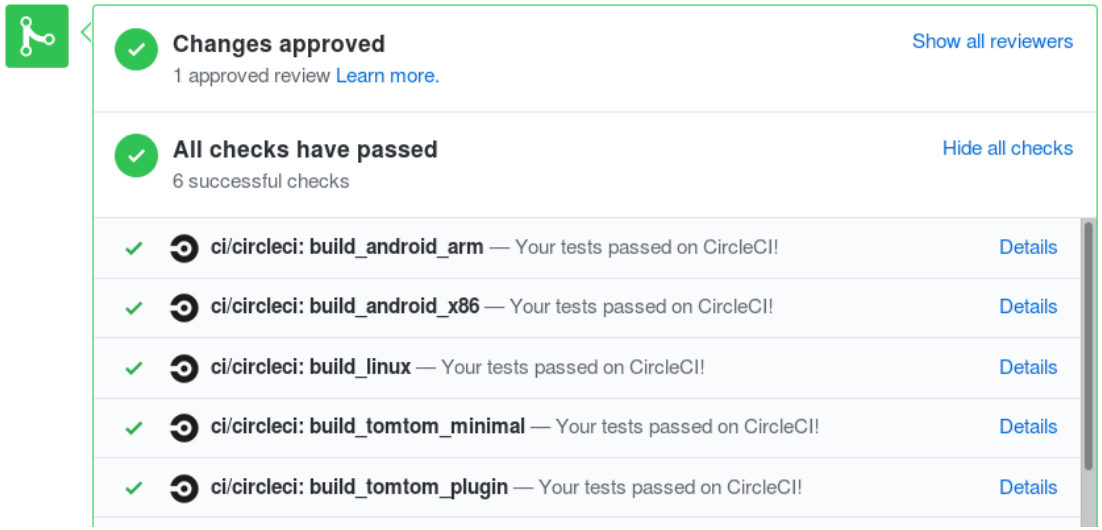
- Build jobs focussed on relevant artifacts:
  - .apk for Android
  - .exe for Windows builds
  - system images for raspberry and Tomtom
  - .rpm for Sailfish builds
- Same process used for:
  - Building
  - Signing
  - Automatically uploading

Outline

- Introduction
- Build-Infrastructure
- **Continuous Integration**
- Device Farm
- Appium
- Conclusion and Future Work

Continuous Integration

- Continuous Integration on circleci
- Integrated with Github
- Benefits:
  - direct testing of pull requests
  - Reduction of scope of branches to one specific thing
  - easier reviewing and merging



Outline

- Introduction
- Build-Infrastructure
- Continuous Integration
- **Device Farm**
- Appium
- Conclusion and Future Work

Device Farm

Challenges for OpenSource smartphone app projects

- Large number of supported mobile devices demand extensive testing on many different platforms
- Android has a large version fragmentation
- iOS requires Mac-Computers which are not always available

Device Farm

Several companies systems offer a device farms

- Online interface to a larger available set of mobile phones for device and platform specific testing. (build for platform or device broken?)
- Run GUI-based test (usability same on different devices and platform?)
- Run-time performance evaluation on different devices

Device Farm

Available device farms:

- Amazon Device Farm
- Google Test Lab
- Xamarin Test Cloud
- Kobiton
- Experitest
- Sauce Labs
- OpenSTF

Outline

- Introduction
- Build-Infrastructure
- Continuous Integration
- Device Farm
- **Appium**
- Conclusion and Future Work

Appium

- GUI testing framework for Android, iOS, Windows Mobile and Web based apps
- Partially based on Selenium and Node.js
- Clients for different programming languages, e.g. Python, Java, PHP, Python, Ruby or C#
- Released under Apache License
- Supported on AWS Device Farm
- Available at: http://appium.io/

Appium

- Initial implementation on AWS Device Farm and local device
- First simple tests with startup and initialization of app, e.g. downloading of map and selection of text-to-speech-system

Appium

Example Source code:

```
self.driver.start_activity('org.navitproject.navit',
'org.navitproject.navit.Navit')
el = self.driver.find_element_by_android_uiautomator('new
       UiSelector().textContains("Google")')
el.click()
el = self.driver.find_element_by_android_uiautomator('new
       UiSelector().textContains("Europe")')
action = TouchAction(self.driver)
action.press(el).release().perform()
while True:
     sleep(5)
     try:
             el = self.driver.find_element_by_android_uiautomator('new
                  UiSelector().textContains("downloading")')
     except Exception as e:
              break
```

Appium

Example Video:

Video Demo

Outline

- Introduction
- Build-Infrastructure
- Continuous Integration
- Device Farm
- Appium
- **Conclusion and Future Work**

Conclusion and Future Work

- Android build on circleci
- Distributed via Google-Play and F-Droid
- Current testing during compile phase
- GUI testing using Appium
- Initial tests on AWS Device Farm
- Work towards tests on own device farm

You have reached your destination in one slide.

Contacts

Homepage:	www.navit-project.org
Forum:	forum.navit-project.org
Wiki:	wiki.navit-project.org
Bugtracker:	trac.navit-project.org
IRC:	#navit on freenode.net
Facebook:	www.facebook.com/NavitProject
Twitter:	www.twitter.com/NavitProject

You have reached your destination now.