# MESS

***Release 20190314.16***

**Mar 14, 2019**

# Contents

**Modern Engineering** Software-engineering is a juvenile profession, with many new insights.

Most are recently *discovered* in generic software development; like 'the web'. And written in a language (both computer and human), that is very contrasting with the technology-talk of typical "RealTime/Embedded" engineering. And so, often rejected or not seen as relevant.

Although their examples are too dissimular in many casses, the concepts can be useful. There is no valid reason not to incorporate their modern approach of software-engineering into *our* process.

**Sovereign Software** All **unmanaged**, *background* software enabling modern life!

- Traditional RealTime/Embedded software & their big successors
- Compilers, Kernels, Drivers, code-analysers, ...
- Routers, PLCs, EMUs, the TCP/IP-stack, ...

Traditionally, that "small" software was called "RealTime" and/or "Embedded". Currently most of that software isn't *'small'* anymore, nor *'embedded (in a product)'*. Furthermore, all *normal* software appears to be realtime; mostly due hardware-speedup.

And, there is a lot of software, like (kernel)-drivers, compilers and SW-tools which isn't embedded, nor realtime; but should have the quality that is equivalent to that old- school engineering. As everybody depends on it – directly (sw-engineering) or indirectly (a bug in a compiler will effect all end-users!).

CHAPTER 1

# Table of Content

## 1.1 ReadMe

**Engels & Dutch**

- This side is partly in Dutch.
- En gedeeltelijk in het Engels

**Modern Engineering** Software-engineering is a juvenile profession, with many new insights.

Most are recently *discovered* in generic software development; like 'the web'. And written in a language (both computer and human), that is very contrasting with the technology-talk of typical "RealTime/Embedded" engineering. And so, often rejected or not seen as relevant.

Although their examples are too dissimular in many casses, the concepts can be useful. There is no valid reason not to incorporate their modern approach of software-engineering into *our* process.

**Sovereign Software** All **unmanaged**, *background* software enabling modern life!

- Traditional RealTime/Embedded software & their big successors
- Compilers, Kernels, Drivers, code-analysers, ...
- Routers, PLCs, EMUs, the TCP/IP-stack, ...

Traditionally, that "small" software was called "RealTime" and/or "Embedded". Currently most of that software isn't *'small'* anymore, nor *'embedded (in a product)'*. Furthermore, all *normal* software appears to be realtime; mostly due hardware-speedup.

And, there is a lot of software, like (kernel)-drivers, compilers and SW-tools which isn't embedded, nor realtime; but should have the quality that is equivalent to that old- school engineering. As everybody depends on it – directly (sw-engineering) or indirectly (a bug in a compiler will effect all end-users!).

### 1.1.1 Copyright

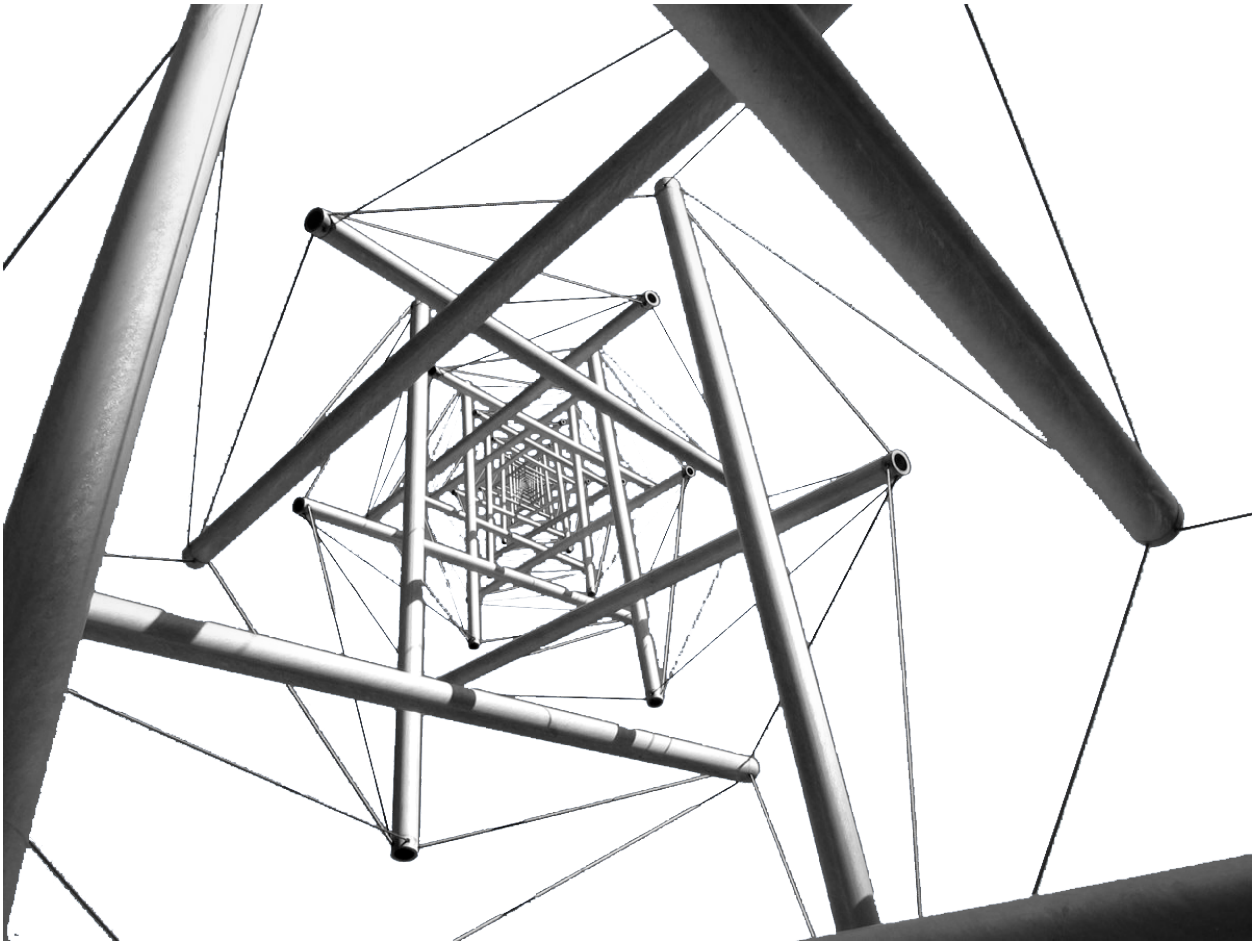
All files and ideas are (C) Albert Mietus. You may:

- Read & study them
- Use the ideas to improve your skills

Please use the *disqus* sections (below) to give your feedback and opinions.

–Albert Mietus

#### Tensegrity, as inspiration



Tensegrity is a synthesis of the names 'tensional' and 'integrity'. It is based on *"teamwork"* of tension and compression forces. Although the image may look confusing, these structures are really very simple. All you need are some poles, some cable and good engineering. This can result is a beautiful 'tensegrity-tower' where the poles almost float in air; as shown above

It is also a well-known architectural principle for skyscrapers!

For me, it is also a inspiration for Software-Engineering: It should be based of teamwork: a synthesis of creative and verifying people. Together with a methodical way-of-working the amplify each other. Then, the sky becomes a limit, which is easy!

## 1.2 PyMESS: MESS with python

### 1.2.1 Training snippets

#### dPID: A Python 'homework' exercise

**status** Alfa

This is an optional exercise for the python-3 workshops: program a **d**iscrete **PID**-controller.

A basic `class definition` is given; which has to be tested and implemented. By starting with the *test-part*, which is advisable anyhow (the TDD approach), the exercise starts simple.

A few *test-examples* are also given. This file can be used as *'template'* to write your own tests.

#### dPID articles

#### dPID: The dPID class

**status** RC-1.0

This article shows the `dpid.PID` documentation, as specified in *the python file*.

It specifies the *interface* to a discrete-PID-controller. Study it to understand how the class should be used. Then start writing python code to test it.

**Attention:** Its' a **python-coding** exercise

When part of this controller is 100% clear, just assume it is working correctly. And fill in the details as needed. Use that as base for your test-code.

- At least, eventually, the class and the code are consistent. So, future changes will not invalidate current (assumed) behaviour; without notice.
- During normal development, such details should be incorporated into the doc-string; possible after discussion and/or approval
- For the 'homework-goal' the exact working isn't that relevant. Fill in details as needed; **focus on writing python-code**!

**class** dpid.**dPID**(*P*, *I*, *D*, *min_result=None*, *max_result=None*)

A simple discrete-PID controller, as an exercise.

This PID-controller can be initialised with constantes for `P`, `I` and `D`; which can't be changed afterwards. Optional, a minimum and maximum output value can be given; both during initialisation, and later.

The controller has two **inputs**: `setpoint()` and `measured()`, and one **output**: `result()`. Those inputs can be set/updated independently. Similarly, the `result()` can be read at any-moment. As the controller will *remember* the timestamp a values changes (and knows when the result is read), it will always give the correct output. Thus, that output value does depend on the timestamp it is requested!

The `setpoint()` is considered as a step-function: between two changes, it will remain the last value. The `measured()` value however should be considered as a continuously linear-changing value. So, between two updates of this value, the dPID-controller will interpolate linearly.

When a `result()` is read during such a period; the PID-controller can't predict the next-measured-value, however. Therefor, it will (for that single read) assume the measured-value is the same as last-time.

When a maximum and/or minimum value is set, the *result()* will be clipped to that value when needed. Without a min/max, the *result()* is unlimited.

---

**Hint:** As this class is part of an exercise; no implementation is given.

During the training one should update this file to implement the class **without** changing the interface.

All (numeric) input & output values are either integers or floats.

---

**setpoint**(*sp*)
Set the setpoint: a numeric value.

**measured**(*value*)
Give the controller an update on the actual *measured* (or simulated) process-value.

The controller will assume a linear progression between the last update and the current one

**result**()
Return the actual result value

**set_min_max**(*min_result=None*, *max_result=None*)
Change the minimum and/or maximal result value. Used to clip the *result()*

### dPID: Exercise

**status** pre-alpha

### TDD-approach

### Test first

First, write some test-files with test-functions that verify the correct working of the *dpid.dPID* class as specified. Depending on your knowledge of PID-controllers these test may (functionally) vary. The primary goal here is not to program (including testing) a great PID-controller; but to practice your python-skills.

---

**Hint:** time-dependent

The *dpid.dPID* class is discrete; it only calculates the *dpid.dPID.result()* when requested. This implies the (return) value of *dpid.dPID.result()* will depend on when it is called (relative to the other methods).

- So, timing may be relavant in your (test)-code. One can use `time.sleep` to controll that. Usa a floating-point parameter for sub-second resolution. Typically, the function is accurate in the milli-seconds range.
- Compare using a small *MARGIN*, to allow derivations due e.g. timing. See the *examples*.

**See also:**

Convenient python functions

- https://docs.python.org/3.5/library/time.html?highlight=sleep#time.sleep
- https://docs.python.org/3.5/library/functions.html?highlight=abs#abs

---

### Instructions

- Use the pytest framework, to shorten test-code.
  - *Pytest introduction* is a summary with everything you need for this exercise.
  - You can use *Some (py)test examples* as *template* for your own files.
- Start by running `pytest` as shown.
  - Reproduce the shown output first, before adding your own files.
  - Remember; the output text will differ slightly; for paths, dates etc.
- Add a (one) new file: `test_<something>.py`.
  - Copy the start of the file (above the first function); update the copyright.
  - Write a single test-function:
    - Create a *`dpid.dPID`* instance with known, **simple** `P`, `I` and `D` settings.
    - Give it a *`setpoint()`* and *`measured()`* value.
    - Request a *`result()`*.
    - Assert the returned value is (almost) equal to the pre-computed number.
- Run `pytest` again.
  - It should run both the existing example, and the new test-file.
  - The test should fail! As the *empty* class always returns `0`, that is easy.
  - Watch for *syntax* and other errors! All, but AssertionError, should be resolved in your code
- Repeat, either by adding a test-function to that file, or adding more test-files.
  - When needed, you can add auxiliary functions; just don't use the *test*-phrase in its name.
  - Or continue first with the implementation part. And add more test later (for other functions).
  - Each test should be a little more complicated as the existing onces.
  - Or better: start with the trivial once. Then the almost-trivial, etc.
    - Start testing a "*P-only*" PID-controller
    - Then an "*I-only*", then a "*D-only*". After which you test a simple combination
    - etc.

### Code second

When a part of the functionality is tested (or at least: there is test-code for), you can start implementing the *`dPID`* class. Keep is simple. The only objective is to **make one failing test pass**.

### And improve (refactor)

### dPID: The code

**status** RC-1.0

The code of the test-examples and the (empty) *dpid.dPID* class are shown here. They are exactly as the python-files; but for the highlighting.

### Some (py)test examples

```
# Copyright (C) 2017: ALbert Mietus, SoftwareBeterMaken
# Part of my MESS project
# Dropjes licencie: Beloon me met dropjes naar nuttigheid

import pytest

from logging import getLogger
logger = getLogger(__name__)

from dpid import dPID

def test_P():
    MARGIN  = 0.5

    c = dPID(1,0,0)

    c.setpoint(10.0)
    c.measured(10.0)
    out = c.result()

    assert (-1*MARGIN) < out < MARGIN, "result (%s) should be close to zero (MARGIN=
↪%s)" % (out, MARGIN)

def test_clip():
    c = dPID(1,2,3)

    c.set_min_max(min_result=10)
    c.set_min_max(max_result=10)

    for sp in range(-100,100,10):
        c.setpoint(sp)
        c.measured(0)

        got = c.result()
        assert got == 10, "Both min and max are clipped to 10; so result should be 10!
↪. But it is: %s" % c.result()
```

### The class (empty)

```
# Copyright (C) 2017: ALbert Mietus, SoftwareBeterMaken
# Part of my MESS project
# Dropjes licencie: Beloon me met dropjes naar nuttigheid


from logging import getLogger
logger = getLogger(__name__)

class dPID:
    """A simple discrete-PID controller, as an exercise.
```

(continues on next page)

(continued from previous page)

```

    This PID-controller can be initialised with constantes for ``P``, ``I`` and ``D``;
    which can't be changed afterwards.  Optional, a minimum and maximum
    output value can be given; both during initialisation, and later.

    The controller has two **inputs**: :meth:`.setpoint` and
    :meth:`.measured`, and one **output**: :meth:`.result`. Those inputs can
    be set/updated independently. Similarly, the :meth:`.result` can be read
    at any-moment. As the controller will *remember* the timestamp a values
    changes (and knows when the result is read), it will always give the
    correct output. Thus, that output value does depend on the timestamp it
    is requested!

    The :meth:`.setpoint` is considered as a step-function: between two
    changes, it will remain the last value. The :meth:`.measured` value
    however should be considered as a continuously linear-changing value. So,
    between two updates of this value, the dPID-controller will interpolate
    linearly.

    When a :meth:`.result` is read during such a period; the PID-controller can't
    predict the next-measured-value, however. Therefor, it will (for that
    single read) assume the measured-value is the same as last-time.

    When a maximum and/or minimum value is set, the :meth:`.result` will be
    clipped to that value when needed. Without a min/max, the :meth:`.result` is
    unlimited.


    .. hint:: As this class is part of an exercise; no implementation is given.

       During the training one should update this file to implement the class
       **without** changing the interface.

       All (numeric) input & output values are either integers or floats.

    """



    def __init__(self, P,I,D, min_result=None, max_result=None): pass

    def setpoint(self, sp):
        """Set the setpoint: a numeric value."""

    def measured(self, value):
        """Give the controller an update on the actual *measured* (or simulated)␣
↪process-value.

        The controller will assume a linear progression between the last update and␣
↪the current one
        """

    def result(self):
        """Return the actual result value"""
        return 0.0 # XXX

    def set_min_max(self, min_result=None, max_result=None):
```

(continues on next page)

(continued from previous page)

```
66          """Change the minimum and/or maximal result value. Used to clip the :meth:`.
↪result`"""
```

## Downloads

You can download these files directly from bitbucket

- https://bitbucket.org/ALbert_Mietus/mess/raw/default/pyMESS/training/dPID/test_examples.py
- https://bitbucket.org/ALbert_Mietus/mess/raw/default/pyMESS/training/dPID/dpid.py

## Pytest introduction

**status** Beta

By using pytest, it becomes simple to run one, several or all test-functions. It has many advanced features, which are not needed for this exercise; but feel free to visit the website.

Pytest uses *autodiscovery* to find all tests. This makes all test-scripts a lot shorter (and easier to maintain), as the *"main-trick"* isn't needed in all those files.

Without pytest all test-files should have a section like:

```
if __name__ == "__main__":
   test_P()
   test_clip()
   ...
   # list ALL your test here!
```

Effectively, pytest will automatically discover all test-functions; and execute them as-if that section is added, with all test-function listed (in file-order).

## Example

- Installation of pytest is trivial; use:

```
[Albert@pyMESS:] % pip install pytest
```

- Running all tests (in 1 directory) is trivial too:

```
[Albert@pyMESS:../dPID] % pytest
============================================================ test session starts␣
↪============================================================
platform darwin -- Python 3.4.1, pytest-3.0.4, py-1.4.31, pluggy-0.4.0
rootdir: /Users/albert/work/MESS,hg/pyMESS/training/dPID/dPID, inifile:
collected 2 items

test_examples.py .F

================================================================= FAILURES␣
↪==================================================================
_____________________________________________________________ test_clip ____________
↪_____________________________________________________
```

(continues on next page)

(continued from previous page)

```
    def test_clip():
        c = dPID(1,2,3)

        c.set_min_max(min_result=10)
        c.set_min_max(max_result=10)

        for sp in range(-100,100,10):
            c.setpoint(sp)
            c.measured(0)

            got = c.result()
>           assert got == 10, "Both min and max are clipped to 10; so result␣
↪should be 10!. But it is: %s" % c.result()
E           AssertionError: Both min and max are clipped to 10; so result should␣
↪be 10!. But it is: 0.0
E           assert 0.0 == 10

test_examples.py:33: AssertionError
================================================== 1 failed, 1 passed in 0.09␣
↪seconds ==================================================
```

**Note:** expect *AssertionError*s (**ONLY**)

- As the class isn't implemented, one should expect *Asserts* during those (initial) runs.
- Make sure you find *AssertionError*s only; no syntax-errors etc! They denote mistakes in your code!

- The used test-file (test_examples.py) can be found *here*

### Conventions

To make this (*autodiscovery*) possible, one has to fullfil a few conventions:

1. All (python) files containing test-functions, should start with `test_`
   - Alternative: end with `_test.py`
   - Typically, *I* use the prefix for black-box and glass-box tests. And the suffix for white-box tests.
2. All test-functions should have a name starting with `test_`
   - No other function should *not* use that prefix!
3. Test-functions are called without arguments
   - We don't use/need fixtures here; which look like function-parameters. So, define all test-functions without parameters!

### OK or NOK: Assert on failure

Every test should result in a single-bit of information: OK nor Not-OK. Sometimes it may be useful to log (print) intermediate results; that can't replace the OK/NOK bit however.

With pytest this is easy: use the `assert` *statement*!

Typically a test ends with an assert. However, it's perfectly normal to have many asserts in one test-function; each one acts as a kind of sub-test. When a test succeeds hardly any output is generated; preventing cluttering of the test-reports.

When the first assert-expression results in `False` the test Fails. Then that AssertionError is show with some context. Giving the programmer feedback on which test fails and why.

> **Warning:** Assert is NOT a function
>
> In python `assert` is a keyword with one or two expressions.
>
> Don't use it as a function; which is a common (starters) mistake. Then, it is read as a single expression: a tuple with two elements. Which is always `True`. So the `assert` never fails!
>
> Typically, the second expression is a string explaining what is expected. And so, documents that part of the test.

**See also:**

Links

- https://pytest.readthedocs.io/
- https://en.wikipedia.org/wiki/PID_controller
- https://en.wikipedia.org/wiki/Test-driven_development

## 1.3 Software competency

### 1.3.1 HTLD: *HighTech Linux Delivery*

Here, you will find some blog-style articles about HTLD; An service Sogeti.HighTech (NL) is going to offer.

#### De Embedded Linux Expert bestaat niet

**reading-time** 3 minuten

Regelmatig krijg ik 'Embedded Linux Experts' aangeboden; bijvoorbeeld voor het **HTLD**-team (*HighTech Linux Delivery*). Vaak zijn goede mensen, met Linux ervaring; maar toch niet de mensen die ik zoek. Het blijkt erg lastig voor niet direct betrokken, om de juiste experts te spotten. Daarom een paar hints waarmee je "de" Embedded Linux Expert herkent. Immers, alleen het woordje 'Linux' op een CV is onvoldoende!

Zo'n 15 jaar geleden was ik één van de mensen die "**Embedded Linux**" in Nederland introduceerden. Via diverse presentaties en artikelen resulteerde dat in "moderne" expertise. Ook was er een tool: EQSL; Embedded QuickStart Linux. Immers, voor veel ervaren *realtime/embedded* ontwikkelaars was Linux *toen* heel nieuw, heel groot & complex en vooral *verwarrend*. Met de EQSL-CD konden ze een toch snel een Embedded Linux systeem bouwen. Vanaf boekje lezen, tools en opties selecteren, Linux bouwen, het image installeren tot en met opstarten van een custom embedded Linux, en dat in minder dan een halve dag! Dat was in 2006 een hele prestatie; er waren immers niet veel *Embedded Linux experts*.

Opnieuw ben ik bezig met een Linux service, en dus aan het (uit)bouwen van team van Embedded Linux Experts. Nu met een ander 4-letter acroniem: **HTLD**; *HighTech Linux Delivery*. Er is veel belangstelling, en ook veel professionals willen graag meedoen.

### Wat kan een Linux expert?

Linux wordt nu alom gebruikt en is nog steeds "sexy". Toch is er nog steeds verwarring. Want wanneer ben je expert? Iemand die een standaard distributie kan installeren op een standaard PC, met twee of drie muis-kliks is duidelijk nog geen expert. Maar ga je al meetellen na duizend-uur, of pas na tienduizend? En hoe zit het met iemand met en 10-jaar ervaring, en Linux ervaring?

Er zijn waarschijnlijk meer dan één miljoen experts op gebied van Linux; waarvan duizenden in Nederland. Maar niet al die experts hebben expertise in het hetzelfde stukje Linux! De overgrote meerderheid richt zich op "klassieke IT". Vaak zijn het geen *ontwikkelaars* maar *beheerders*. Erg handig om "high availability" clusters (of tegenwoordig: cloud computers) te configureren; maar wellicht niet bedreven in het ontwikkelen van een Linux-*driver*.

> Een Embedded Linux Expert kan veel meer dan "Linux" gebruiken! Zo moet zij/hij **ook** een ontwikkelaar zijn. En nog veel meer ...

### Wat kan een Linux ontwikkelaar?

Er zijn heel veel ontwikkelaars, die verstand van Linux hebben. En opnieuw zijn ze niet gelijk. Globaal zijn er 3 soorten linux-ontwikkelaars; afhankelijk van wat ze ontwikkelen. De vraag is:

1) Maken ze applicaties die (toevallig, ook) op Linux draaien?
2) Gebruiken ze Linux als ontwikkelomgeving?
3) Of, ontwikkelen ze Linux zelf?

   Ik ben vooral geïnteresseerd in het laatste type. Een Embedded Linux expert moet **ook** de source van Linux zelf beheersen.

De Expert die we zoeken heeft dus verstand van Linux, is ontwikkelaar, en heeft ervaring met de source-code van Linux zelf. Typisch hebben ze ook veel passie voor Linux: het is meer dan werk of een hobby; het is bijna een *life-style*. Een beetje Linux-ontwikkelaar compileert regelmatig zijn eigen kernel, direct van de kernel-sources; liefst met een zelf gebouwde *toolchain*[9]. Vaak vooral omdat het kan, niet omdat het moet; het is immers leuk!

### Wat kan een Embedded Linux Expert?

Is dat complex en ingewikkeld? Ja behoorlijk, voor de meeste mensen. Zoals gezegd, 15 jaar geleden was het zelfs (te) complex voor de meeste, ervaren embedded ontwikkelaars. Inmiddels zijn er duizenden "betaalde "vrijwilligers" die dit met liefde doen. Achter elke 'distributie'[4] zijn Linux Experts druk bezig met het beter maken van de code, en die te compileren; zodat gebruikers die kunnen installeren op hun PC.

U leest het goed: de "PC". De meeste distributies, en de meeste Linux ontwikkelaars richten zich primair op de PC. Dat kan een laptop zijn, of een server; of desnoods een cloud computer.

> Een Embedded Linux Expert kan veel meer ...

Zij/hij heeft **ook** verstand van en ervaring met Embedded Software; dus van "drivers"[6], "BSPs"[7] en "bootloaders"[8]. Een Embedded Linux Embedded kan **ook** "Cross Compileren"[5], met "toolchains"[9] zoals "Buildroot"[11] of "Yocto"[10]; maar ook werken met "GnuMake"[12], want (Embedded) Linux kan niet zonder "Makefiles"[12].

[9] https://en.wikipedia.org/wiki/Toolchain Over de software-tools om software te maken en hun (complexe) relaties.
[4] Er zijn meer dan 500 distributies zoals: Ubuntu, Suse, RedHat, Gentoo, etc! Zie oa https://nl.wikipedia.org/wiki/Linuxdistributie
[6] https://nl.wikipedia.org/wiki/Stuurprogramma Over (device) drivers, ook wel stuurprogramma genoemd.
[7] https://en.wikipedia.org/wiki/Board_support_package Over die software die nodig is om (Linux) op uw eigen computer-board te laten werken.
[8] https://nl.wikipedia.org/wiki/Bootloader Over software die alle software opstart
[5] https://en.wikipedia.org/wiki/Cross_compiler Over cross-compileren (*"XCC"*) en *Canadian Cross Compilers* (Engels).
[11] https://buildroot.org; Buildroot, is een andere toolchain om Embedded Linux te bouwen
[10] https://www.yoctoproject.org; Yocto, is een toolchain om 'custom' Linux te compileren
[12] Linux gebruikt GnuMake; zie https://nl.wikipedia.org/wiki/Make_(computerprogramma) voor Makefile(s)

### Daarmee herken je die expert!

Ja, dat zijn veel complexe termen bij elkaar. Behalve voor de expert die we zoeken; zijn/hij immers een expert!

> Als iemand niet weet wat die termen betekenen, dan is het niet de expert die ik zoek!

Ook verwacht ik van een Embedded Linux Expert, dat zij de "kernel-sources"[1] kan vinden, evenals de "busybox"[2], en **ook** de rest van "Linux-code"[3] kent.

Natuurlijk loop ik nu de kans dat iemand nu die *"aangehaalde termen"* in zijn CV gaat zetten. Maar dan heb ik altijd nog strikvragen zoals: *"Heb je ervaring met de meest gebruikte Embedded Linux processor?"* , om het kaf van het koren te scheiden. Dat is overgens de "ARM"[13]-CPU, die in vrijwel elke smartphone zit. En daarmee heeft bijna iedereen ervaring. Toch?

### Meer weten?

Zou je expert willen worden bestudeer dan vooral die "aangehaalde termen"; hieronder staan een aantal links naar wikipedia; soms is die kennis alleen in het engels beschikbaar. Ook een paar oude publicaties zijn nog beschikbaar.

Daarnaast levert Googlen op "Embedded Linux" ruim 100M-hits op! Kennis zat dus, maar is het te vinden?

> Ook zal ik komende tijd wat meer blogs publiceren over HTLD; zoals een artikel over waarom Linux-drivers vaak duurder worden dan gedacht. Hou deze plek daarom in de gaten.

Natuurlijk mag je ook altijd contact met me opnemen

— Albert.Mietus

---

### Footnotes & Links

Een paar (15 jaar) oude publicaties over Embedded Linux; ze zijn verouderd, maar soms nog verbazend actueel.

- [13 okt 2005] https://bits-chips.nl/artikel/snelle-linux-overstap-begint-bij-toepassing/
- [11 mei 2006] https://bits-chips.nl/artikel/pts-bouwt-opstapje-naar-embedded-linux
- [reprints '12] http://albert.mietus.nl/read.IT/Proponotheek/index.html (reeks van 4)

Een echte Embedded Linux Expert kent deze locaties uit zijn hoofd:

Enkele experts-termen volgens wikipedia:

When you are interested; you can contact me:

— Albert.Mietus; +31 (0)6 526 592 60

## 1.4 BLOG

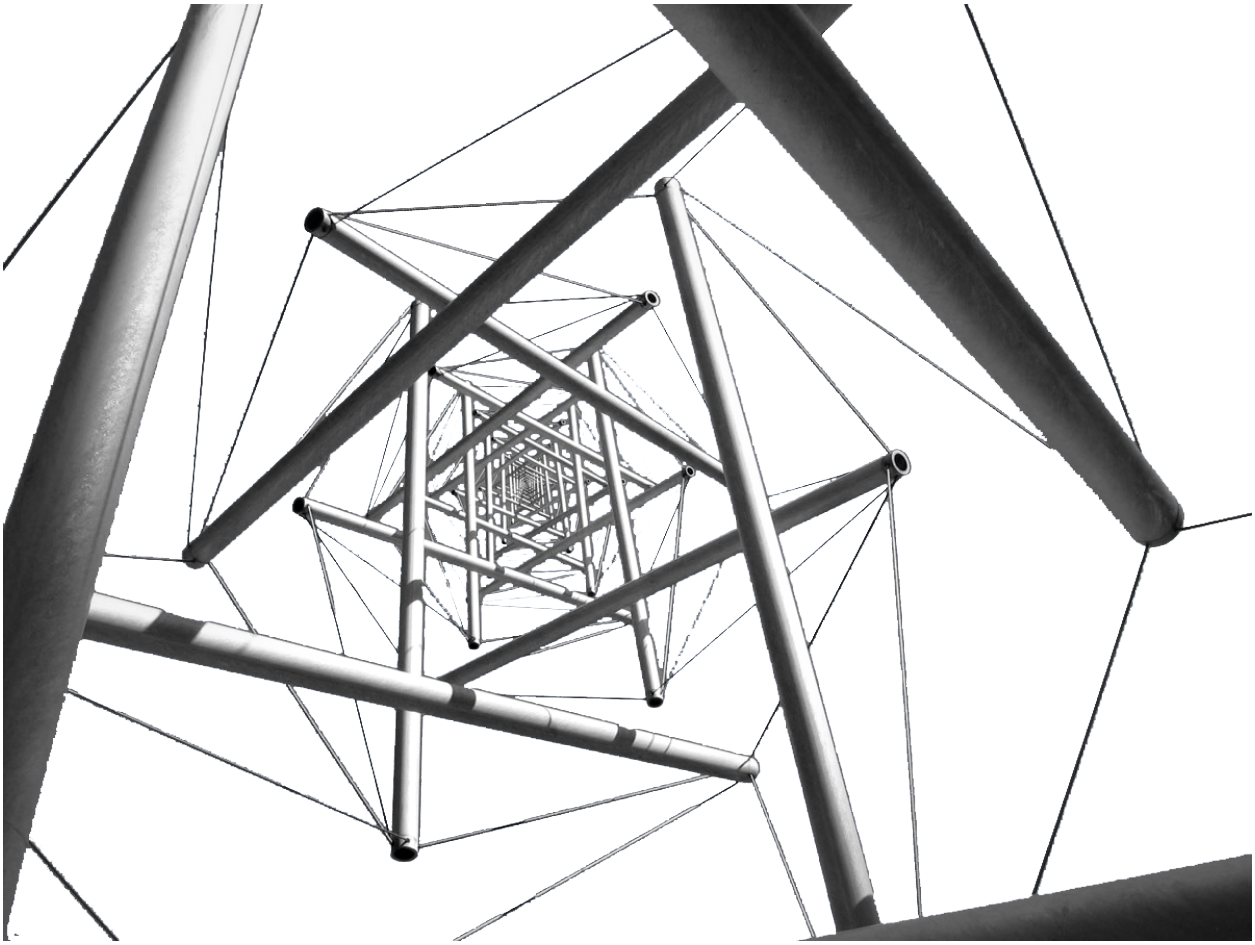
### With postlist

- *De Embedded Linux Expert bestaat niet* (2019/03)

[1] De source van de Linux kernel: https://www.kernel.org
[2] Alle bekende Unix-tools in mini-uitvoering; vooral voor embedded systemen: https://www.busybox.net
[3] Veel andere Linux source code: https://www.gnu.org
[13] De ARM CPU zit niet alleen in (vrijwel) alle smartphones; ook veel andere embedded systemen gebruiken deze. Zie https://nl.wikipedia.org/wiki/ARM-architectuur

CHAPTER 2

# Tensegrity, as inspiration



Tensegrity is a synthesis of the names ‘tensional’ and ‘integrity’. It is based on *“teamwork”* of tension and compression forces. Although the image may look confusing, these structures are really very simple. All you need are some poles,

some cable and good engineering. This can result is a beautiful ‘tensegrity-tower’ where the poles almost float in air; as shown above

It is also a well-known architectural principle for skyscrapers!

For me, it is also a inspiration for Software-Engineering: It should be based of teamwork: a synthesis of creative and verifying people. Together with a methodical way-of-working the amplify each other. Then, the sky becomes a limit, which is easy!

# Index