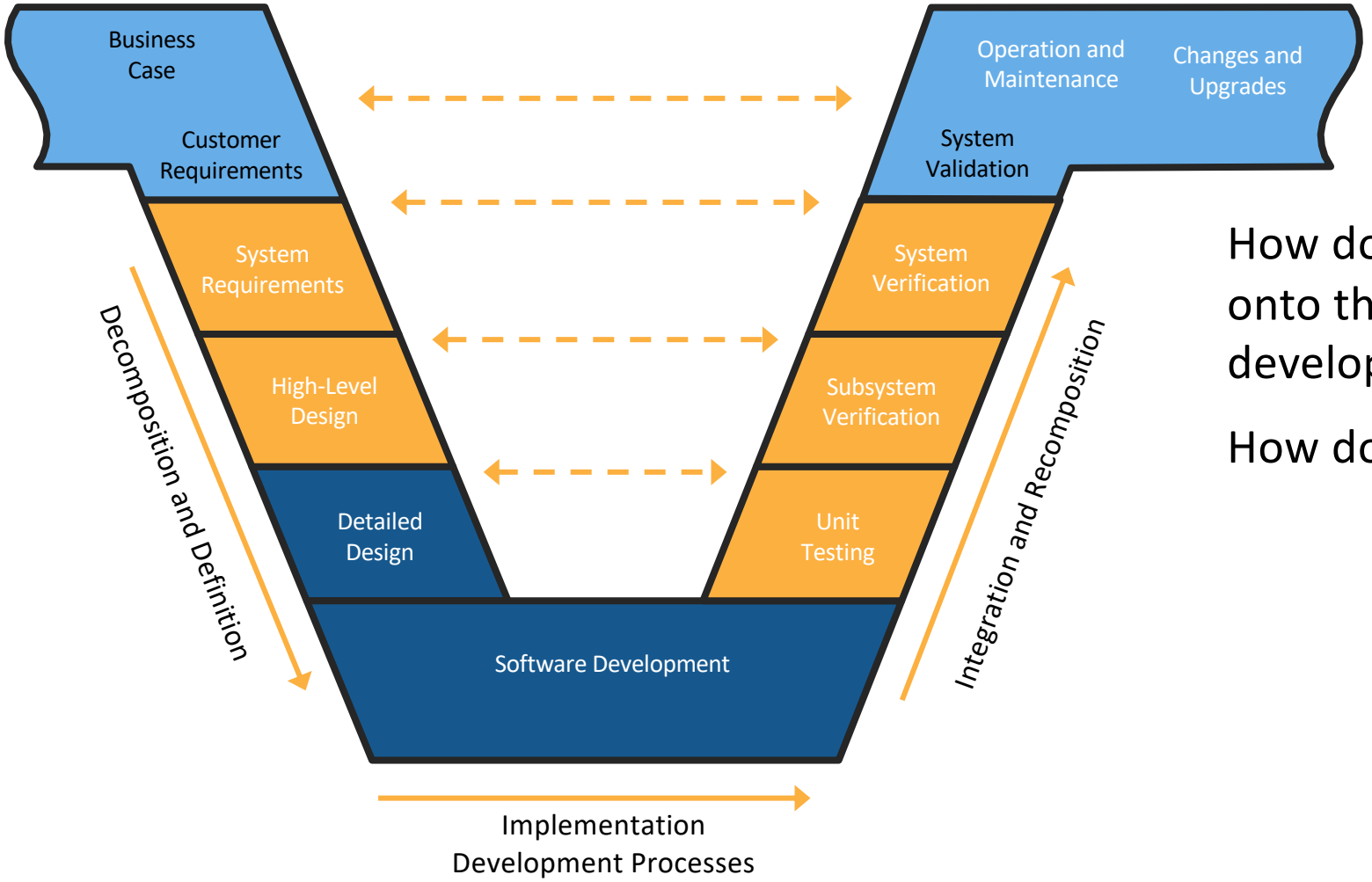
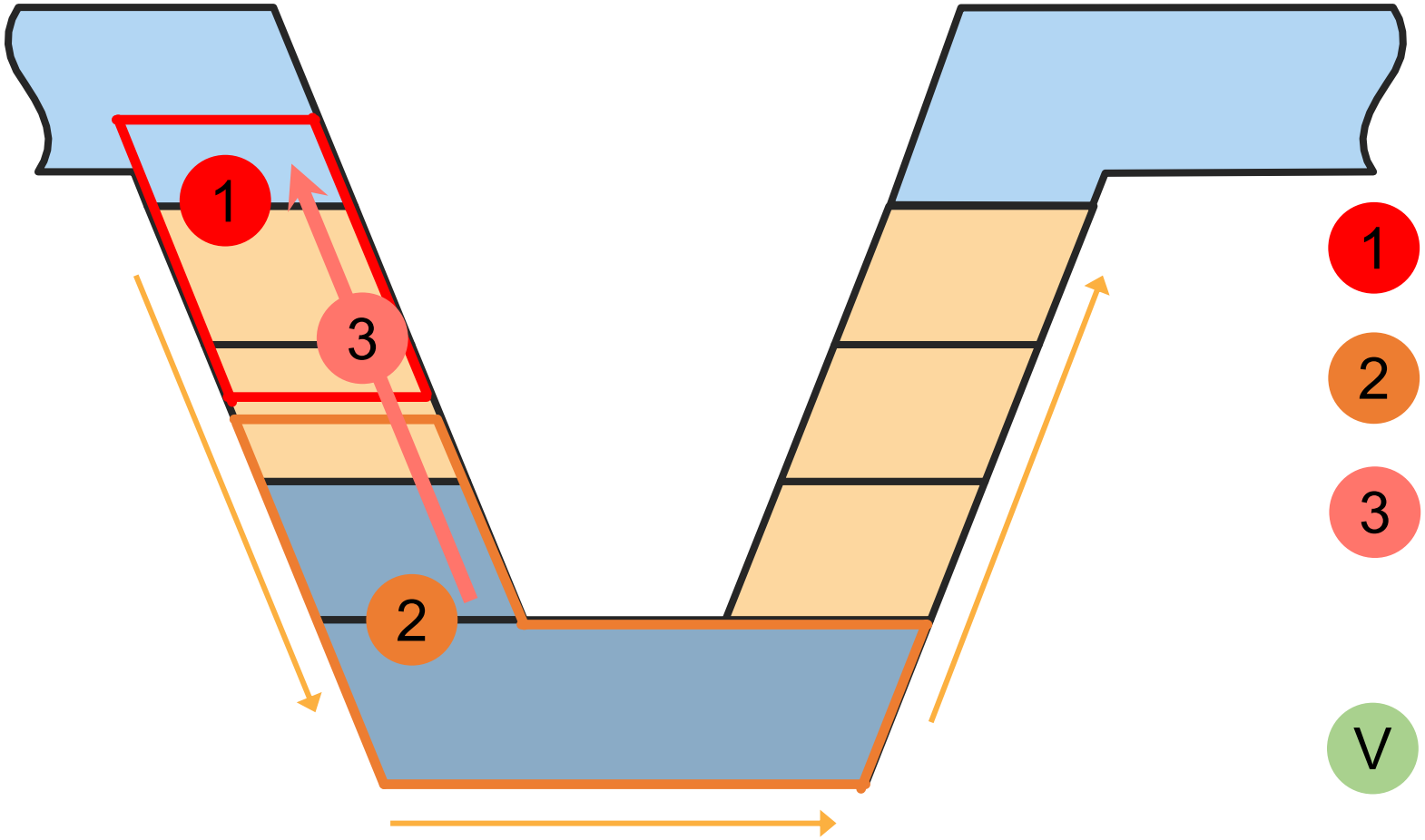
Development Process and Traceability



How do you map this onto the Xen Project development process?

How do you get community buy-in?

What must be upstream: all key inputs …



1 **Requirements documentation**

2 **Design, Architectural and API documentation**

3 **Traceability info:**
Between requirements
Between requirements and other docs
Between requirements and code

V **Validation:**
Can be outside of upstream
Needs a feedback loop to deal with breakage
Needs to trace to requirements (or a written down test case which does)

Is there a tool which fits into a GIT workflow

Doorstop

https://doorstop.readthedocs.io/en/latest/
https://github.com/jacebrowning/doorstop

Installation

Straightforward via

```
$ pip3 install doorstop
```

Start using doorstop within a git tree

Create Document Hierarchy

```
cd docs
doorstop create REQ requirement
doorstop create -p REQ SYS requirements/system
cd ../xen/common
doorstop create -p SYS HLR-common requirements/high-level
doorstop create -p HLR-common LLR-common requirements/low-level
cd ../arch/arm
doorstop create -p SYS HLR-arm requirements/high-level
doorstop create -p HLR-arm LLR-arm requirements/low-level
cd ../..
```

This Creates

```
$ git status
new file:   docs/requirements/.doorstop.yml
new file:   docs/requirements/system/.doorstop.yml
new file:   xen/arch/arm/requirements/high-level/.doorstop.yml
new file:   xen/arch/arm/requirements/low-level/.doorstop.yml
new file:   xen/common/requirements/high-level/.doorstop.yml
new file:   xen/common/requirements/low-level/.doorstop.yml
```

Unique name for document types **per directory**: e.g. HLR-common, HLR-arm

```
$ doorstop
…

REQ
└── SYS
    ├── HLR-common
    │   └── LLR-common
    └── HLR-arm
        └── LLR-arm
```

This Creates

```
$ cat xen/common/requirements/high-level/.doorstop.yml
settings:
  digits: 3
  parent: SYS
  prefix: HLR-common
  sep: ''
```

Human readable config files

Command line interface is similar to git and fairly intuitive

Proliferation of .yml files

Yet another docs source file format: .rst, .markup

Adding Requirements

In xen.git:

```
$ doorstop add REQ --edit -T vi
building tree...
added item: REQ001 (@/docs/requirements/REQ001.yml)
```

```
active: true
derived: false
header: ''
level: 1
links: []
normative: true
ref: ''
reviewed: f8b5fe23e5199f5e03a851f6f9e6f639
text: |
  Dom0less VMs

  Xen shall be able to start Virtual machines in parallel to Dom0
```

Bug? Didn't actually save the changes! Maybe a config issue or specific to Mac

```
$ doorstop review REQ001
```

Adding further requirements

SYS001: Bootloader loads Dom0less VM
The Xen bootloader loads the VM image into memory

SYS002: Text file based config for Dom0less VMs
A text file is used to configure Dom0less VMs

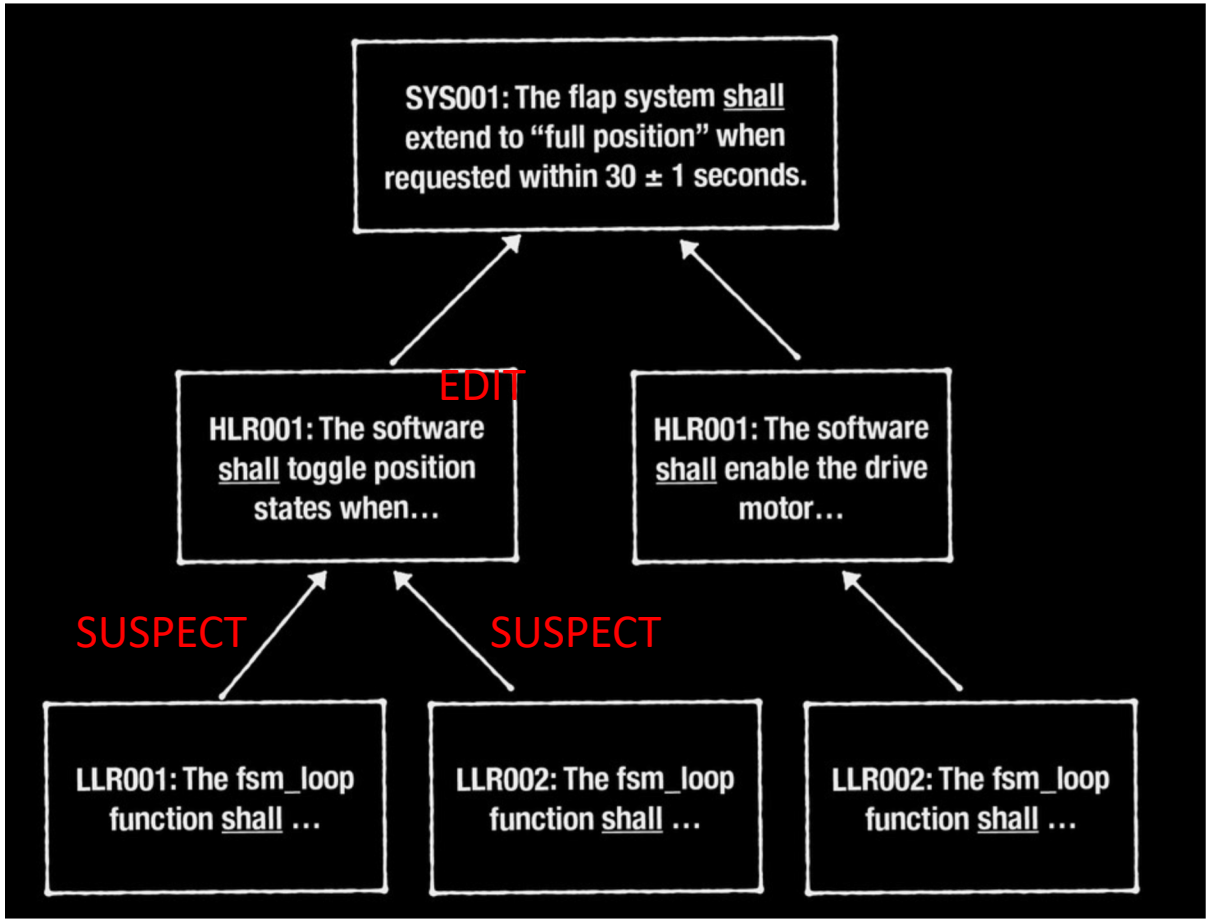
HLR-arm001: Dom0less config via device-tree
Device trees shall be used to configure Dom0less VMs on Arm

```
$ doorstop link SYS001 REQ001
$ doorstop link SYS002 REQ001
$ doorstop link HLR-arm001 SYS002
```

Then, publish a report in html ➜ show this in the file system

```
$ doorstop publish all ../reqv1.html
```

Missing Links, Suspect Links, …



Review and edit the dependent requirements

```
$ doorstop clear LLR001 LLR02
```

BUT:
only does this for links, not references

Referencing Source Code

E.g. a document, function, test case, …

`ref`

External reference. An item may reference an external file or a line in an external file. An external reference is displayed in a published document.

Doorstop will search the project root and it's sub-directories for a filename matching the specified reference. If multiple matching files exist, the first found will be used.

If a file is not found, Doorstop will also search the contents of all text-files in the project root and it's sub-directories. If a line contains the referenced keyword, Doorstop will reference the file and line number where it found the keyword. If the keyword is found in multiple lines or files, the first found will be used.

A file is considered a text-file unless its file extension is listed in `SKIP_EXTS` (settings.py).

The value of this attribute contributes to the fingerprint of the item.

Example: Reference keyword

```
ref: 'TST001'
```

References the filename and line number of a text-file that contains the keyword "TST001".

Example: Reference file

```
ref: 'test-tst001.c'
```

References a file called "test-tst001.c".

References to Source Code

E.g. a document, function, test case, …

```
$ vi docs/requirements/REQ001.yml
ref: 'docs/features/dom0less.pandoc’
$ doorstop review REQ001
$ doorstop
```

Validation suddenly takes a long time (1+ minute)

Searches the tree

Publish a report … crashes
Does not crash with `ref: 'dom0less.pandoc’`

```
$ doorstop publish all ../reqv2.html
```

References to Source Code

E.g. a document, function, test case, …

```
$ vi xen/arch/arm/domain_build.c
/*
 * HLR-arm001.1
 * Some text specific to the requirement
 */
void __init create_domUs(void)
{
…

$ vi xen/arch/arm/requirements/high-level/HLR-arm001.yml
ref: 'HLR-arm001.1’

$ doorstop publish all ../reqv2.html
```

How it shows references to source

Table of Contents

1 Dom0less VMs REQ001

Xen shall be able to start Virtual machines in parallel to Dom0

> `docs/features/dom0less.pandoc`

Child links: SYS001 Bootloader loads Dom0less VM, SYS002 Text file based config for Dom0less VMs

Table of Contents

1.0 Dom0less config via device-tree HLR-arm001

Device trees shall be used to configure Dom0less VMs on Arm

> `xen/arch/arm/domain_build.c` (line 2075)

Parent links: SYS002 Text file based config for Dom0less VMs

Levels and documents

```
cd docs
doorstop create REQ requirement
cd ../xen/common
doorstop create -p REQ HLR-common requirements/high-level
cd ../arch/arm
doorstop create -p REQ HLR-arm requirements/high-level
cd ../..
```

```
REQ
├── HLR-common
└── HLR-arm
```

Document headers and levels

```
$ doorstop add REQ --edit -T vi
Dom0less VMs
Add the ref also
normative: false          # This is essentially just a headline

$ doorstop add REQ --edit -T vi
The Xen bootloader shall load the VM image into memory

$ doorstop add REQ --edit -T vi
A text file shall be used to configure Dom0less VMs

$ doorstop add HLR-arm --edit -T vi
Device trees shall be used to configure Dom0less VMs on Arm Add the
Add the ref also

$ doorstop link HLR-arm001 REQ003
```

Validation and Publication

```
$ doorstop review all
```

```
$ doorstop
building tree...
loading documents...
validating items...
WARNING: REQ: REQ002: no links from child document: HLR-common
WARNING: REQ: REQ002: no links from child document: HLR-arm
WARNING: HLR-common: no items
```

```
 REQ
 │
 ├── HLR-common
 │
 └── HLR-arm
```

```
$ doorstop publish all ../reqv3.html
```

Tree Structure:

```
REQ
|
├── HLR-common
|
└── HLR-arm
```

Published Documents:

- HLR-arm
- HLR-common
- REQ

Item Traceability:

REQ	HLR-common	HLR-arm
REQ002		
REQ003		HLR-arm001

Table of Contents

1.0 Dom0less VMs

1.1 REQ002

The Xen bootloader shall load the VM image into memory

1.2 REQ003

A text file shall be used to configure Dom0less VMs

Child links: HLR-arm001

Table of Contents

1.0 HLR-arm001

Device trees shall be used to configure Dom0less VMs on Arm

> `xen/arch/arm/domain_build.c` (line 1223)

Parent links: REQ003

Making REQ001 not a headline

And make REQ002 & REQ003 children of REQ001

Tree Structure:

Published Documents:

- HLR-arm
- HLR-common
- REQ

Item Traceability:

REQ	HLR-common	HLR-arm
REQ001		

REQ002 does not link to a requirement in HLR-arm
But should show HLR-arm001
Possibly a bug or usage error

Thus: using non-headline REQs to group requirements is of limited use

Summary

Issues: may be bugs, config or usage issues

- Under active development, so may be fixable if bugs

Default's handling not looked into: e.g. -T vi

Outbound references are kind of useless for tracing

The core functionality of creating links between document artifacts seems to work reasonably well

Lots of individual files → not really ideal

- Embed in code and use a script to generate yaml files → breaks workflow
- Extend tool, such that requirement "docs" can be embedded in source files
  - Most code is the GUI/web editor/doc generation
  - E.g. `doorstop add REQ -to mysourcefile.c --edit -T vi`