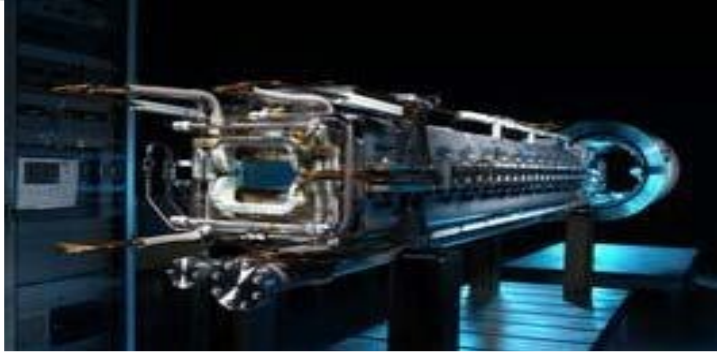




Proposal: Towards a

Common GSI/FAIR Software Development Guideline

Ralph J. Steinhagen

C++ User-Group Meeting

January 24th, 2024

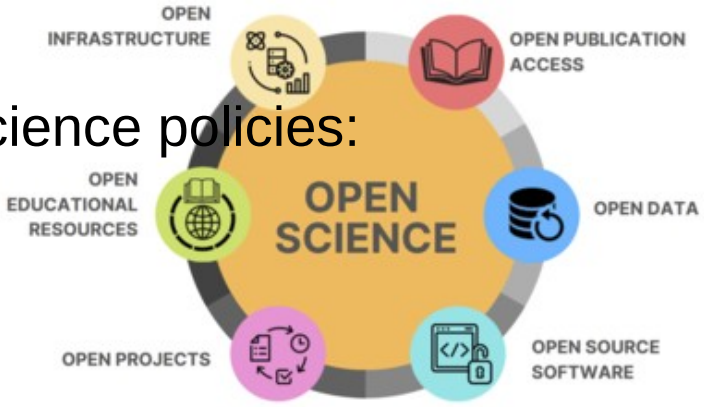
Finland France Germany India Poland Romania Russia Slovenia Sweden UK

GSI HELMHOLTZ ASSOCIATION FAIR

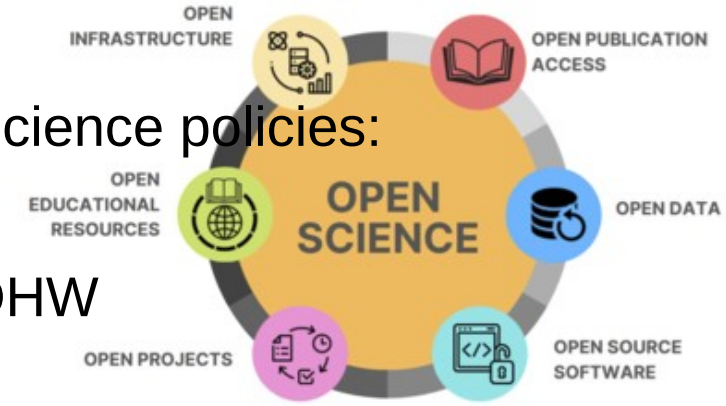
Why a Software Development Guideline?

Why a Software Development Guideline?

not fundamentally ‘new’ – existing GSI/FAIR Open Science policies:



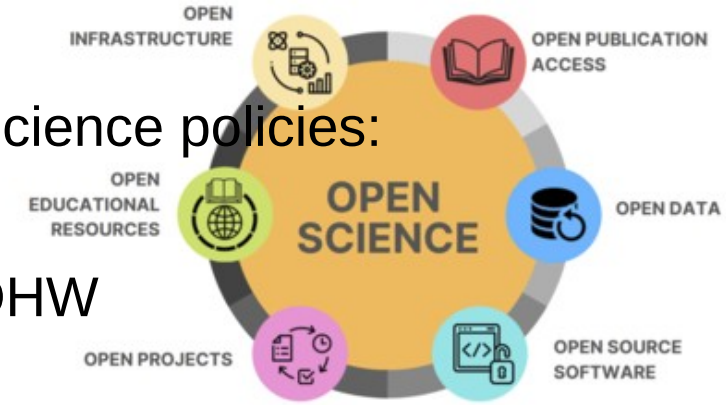
Why a Software Development Guideline?



not fundamentally ‘new’ – existing GSI/FAIR Open Science policies:

- since 2021: Open Source Guidelines
  - GPLv3 → LGPLv3 → Apache 2.0 & CERN OHW
    especially for public-public & public-private collaborations
  - proprietary licenses/development
    require documented evaluation of total-cost-of-ownership & risk for the organisation

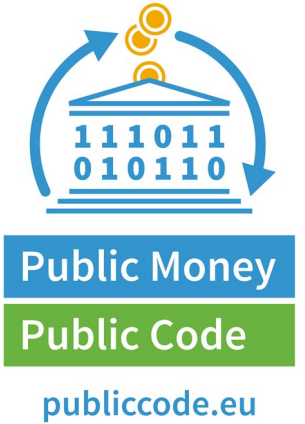
Why a Software Development Guideline?



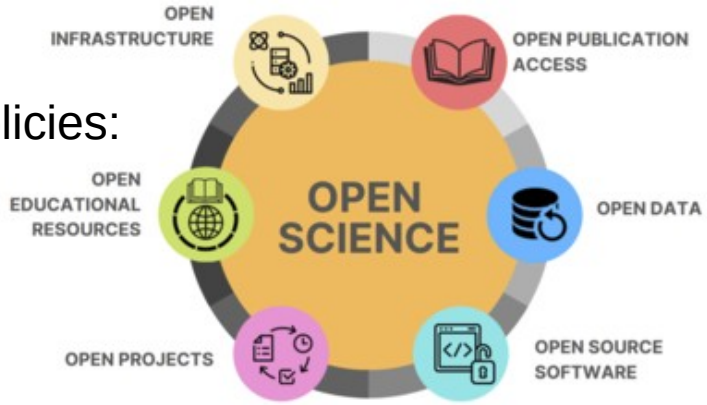
not fundamentally ‘new’ – existing GSI/FAIR Open Science policies:

- since 2021: Open Source Guidelines
  - GPLv3 → LGPLv3 → Apache 2.0 & CERN OHW
    especially for public-public & public-private collaborations
  - proprietary licenses/development
    require documented evaluation of total-cost-of-ownership & risk for the organisation

- since 2023: Open-Science WG Initiative
  GSI/FAIR supporting ‘Public Money? Public Code!’ Campaign
  - new directorate policy:
    favour open-source and document risks related to developments and procurements of new software systems



Why a Software Development Guideline?

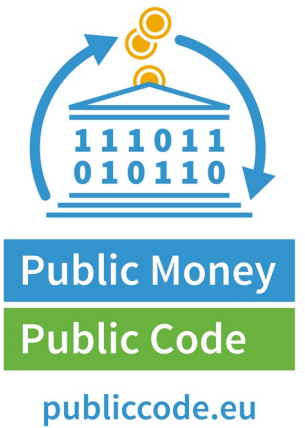


not fundamentally ‘new’ – existing GSI/FAIR Open Science policies:

- since 2021: Open Source Guidelines
  - GPLv3 → LGPLv3 → Apache 2.0 & CERN OHW
    especially for public-public & public-private collaborations
  - proprietary licenses/development
    require documented evaluation of total-cost-of-ownership & risk for the organisation

- since 2023: Open-Science WG Initiative
  GSI/FAIR supporting ‘Public Money? Public Code!’ Campaign
  - new directorate policy:
    favour open-source and document risks related to developments and procurements of new software systems

- 2023/2024: Open-Science WG charged by directorate to draft RSE guidelines for GSI/FAIR
  - refine definition for Open-Hardware/Software policies
  - update licenses (N.B. new AI-regulations and CE certification)
  - Example: DLR Software Engineering Initiative https://rse.dlr.de/



FAIR GSI

Meta-View on Research Software Engineering

Software must be adaptable to frequent changes

FAIR GSI

Meta-View on Research Software Engineering

Software must be adaptable to frequent changes

FAIR GSI

Meta-View on Research Software Engineering

Software must be adaptable to frequent changes

- must provide value and be fit-for-purpose

Meta-View on Research Software Engineering

Software must be adaptable to frequent changes

- must provide value and be fit-for-purpose
- remain easily adaptable to changing requirements

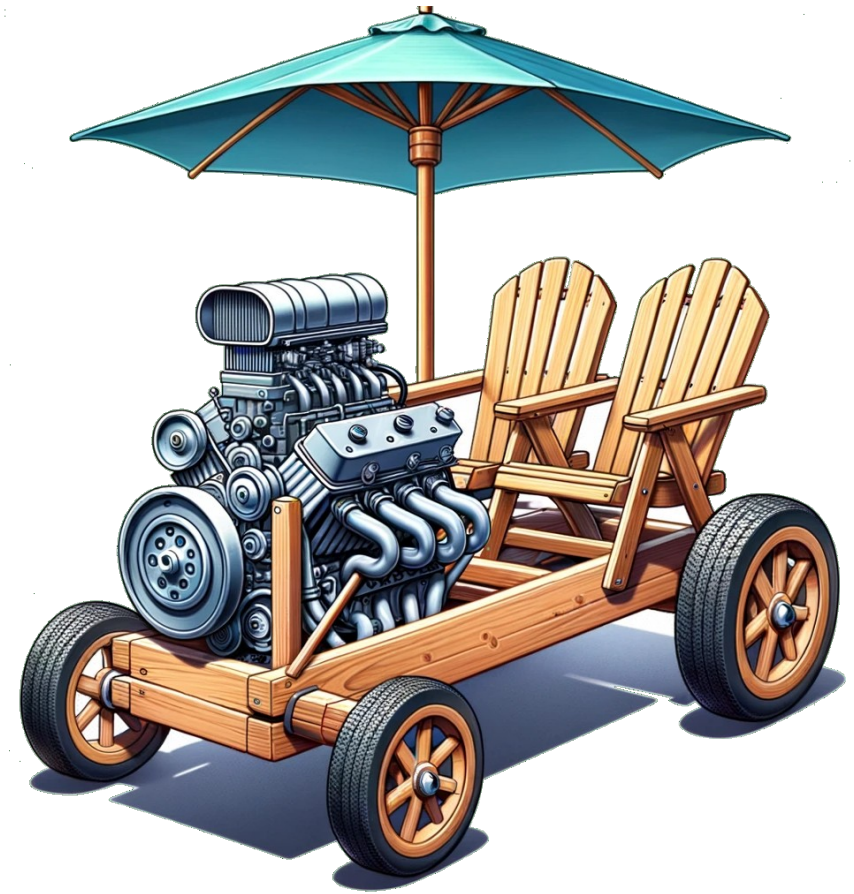
Meta-View on Research Software Engineering

Software must be adaptable to frequent changes

- must provide value and be fit-for-purpose
- remain easily adaptable to changing requirements
- is as much about technology as it is about people

Meta-View on Research Software Engineering

Non issue: numerical algorithms and (usually) industry-wide IT standards

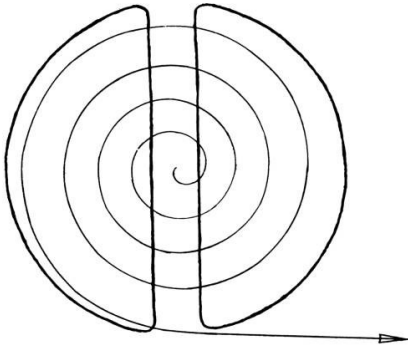


Meta-View on Research Software Engineering

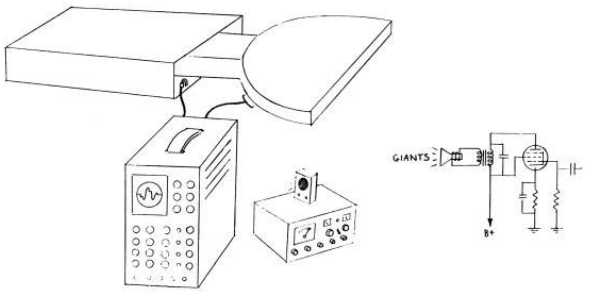
Non issue: numerical algorithms and (usually) industry-wide IT standards

… rather: “Glue-Code”, System-Integration,
and functional evolution

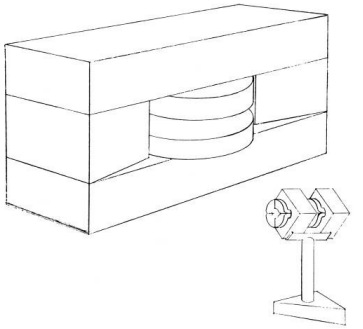
Meta-View on Research Software Engineering – Perspectives



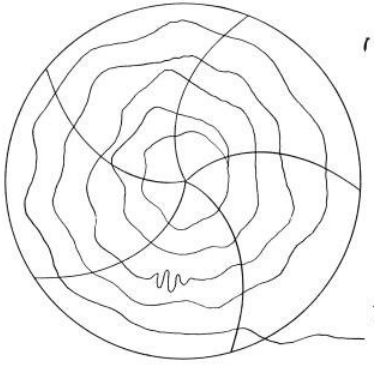
. . . the inventor



. . . the electrical engineer



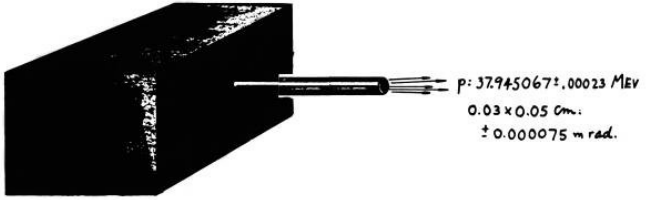
. . . the mechanical engineer



$$I = I_0\left[1+\left(\frac{fr\omega}{c}\right)\cos(3\theta+\delta_0+\delta_1 r)+\left(\frac{fr\omega}{c}\right)^2\cos(5\theta+\delta_3-\delta_5 r^3)+\left(\frac{fr\omega}{c}\right)^3\cos(7\theta+\delta_7-\delta_9 r^3)+ \text{------}\right]\times\left\{\frac{e^{2/5}r^2\ln Z}{1+\left(\frac{a}{r}\right)^{3/4}}\right\}$$

$$\frac{d\phi}{dt}=\left[\sin(\omega t-k\phi)\cdot\sin k\theta-\frac{3}{5}fff_1f_2f_3'\right]\frac{eV_0}{2\pi\omega}$$

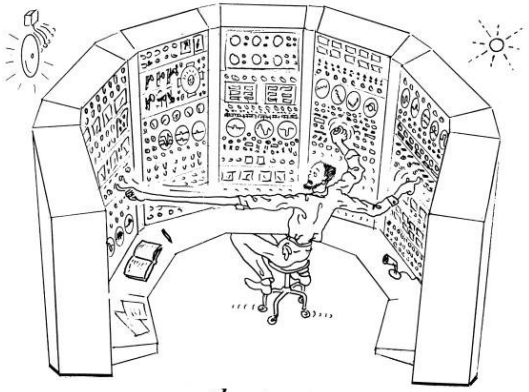
. . . the theoretical physicist



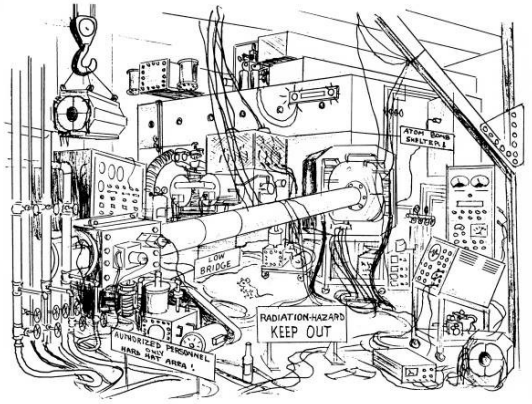
. . . the experimental physicist

FAIR GSI

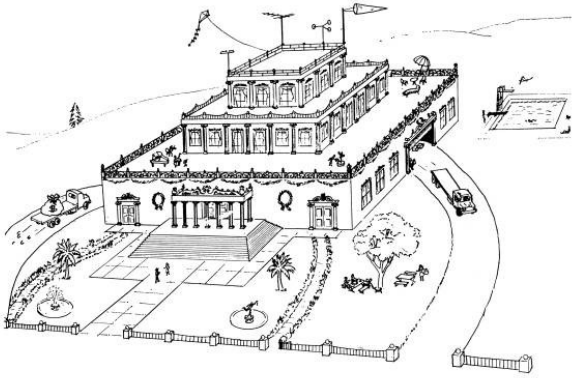
Meta-View on Research Software Engineering – Perspectives



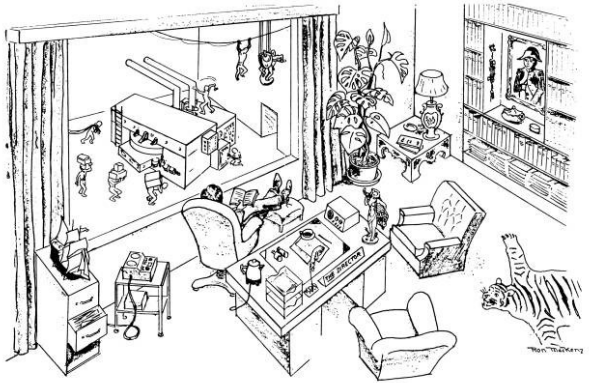
. . . the operator



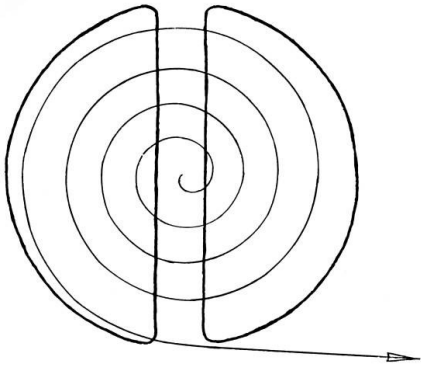
. . . the visitor



. . . the governmental funding agency



. . . the laboratory director

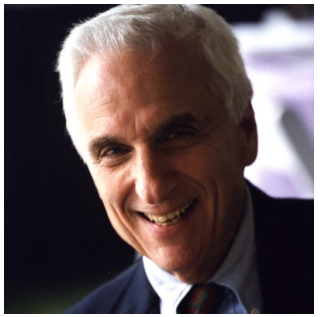


. . . the student

Cartoons: Dave Judd and Ronn MacKenzie

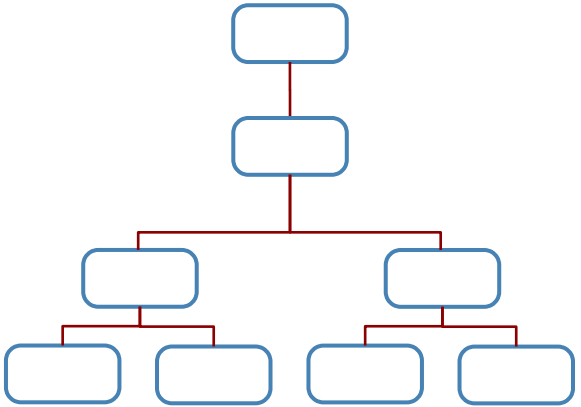
Melvin E. Conway’s Law

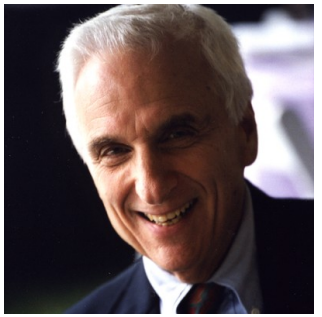
How do Committees Invent? in Datamation, vol. 14, num. 4, pp. 28–31, 1968



Melvin E. Conway’s Law

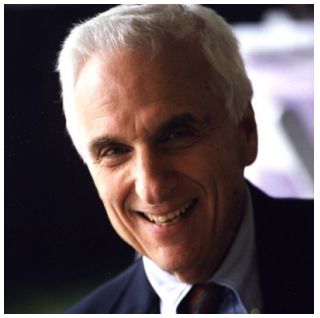
How do Committees Invent? in Datamation, vol. 14, num. 4, pp. 28–31, 1968

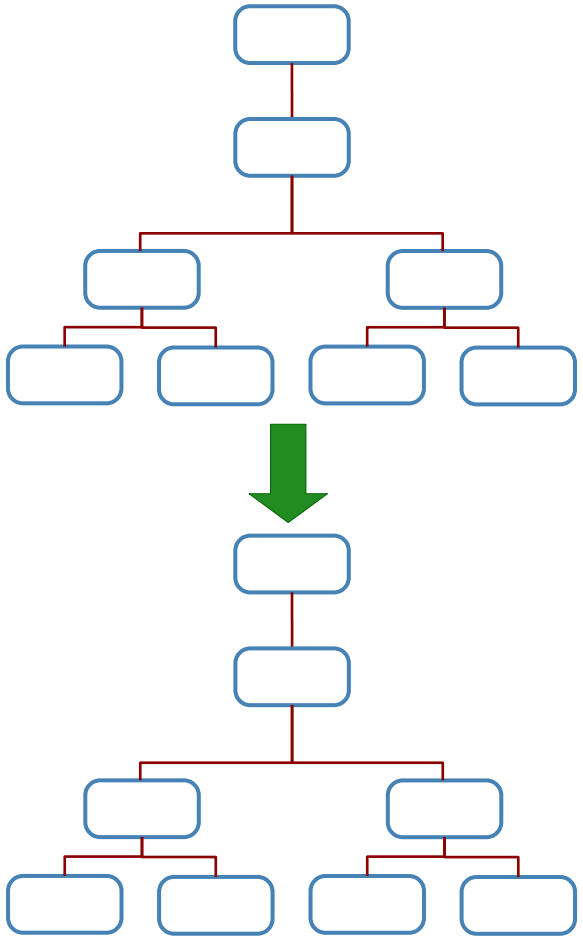




Melvin E. Conway’s Law

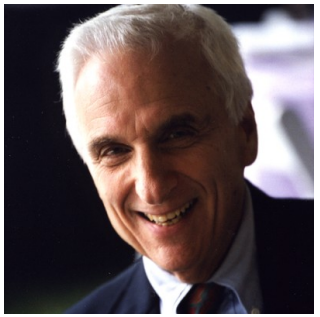
How do Committees Invent? in Datamation, vol. 14, num. 4, pp. 28–31, 1968

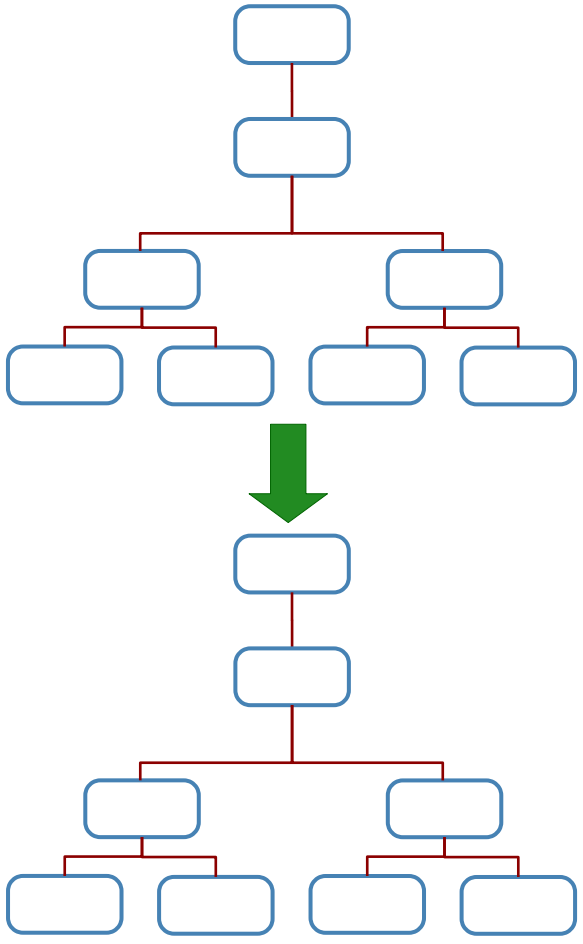


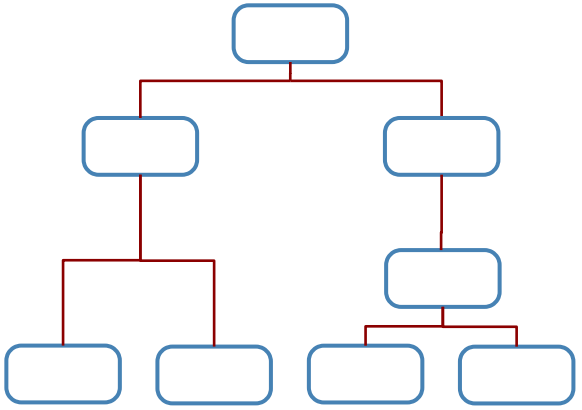


Melvin E. Conway’s Law

How do Committees Invent? in Datamation, vol. 14, num. 4, pp. 28–31, 1968

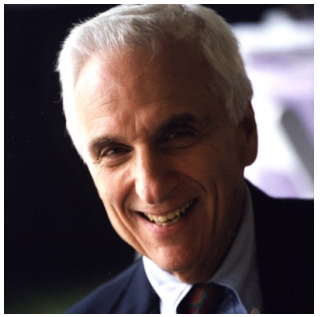


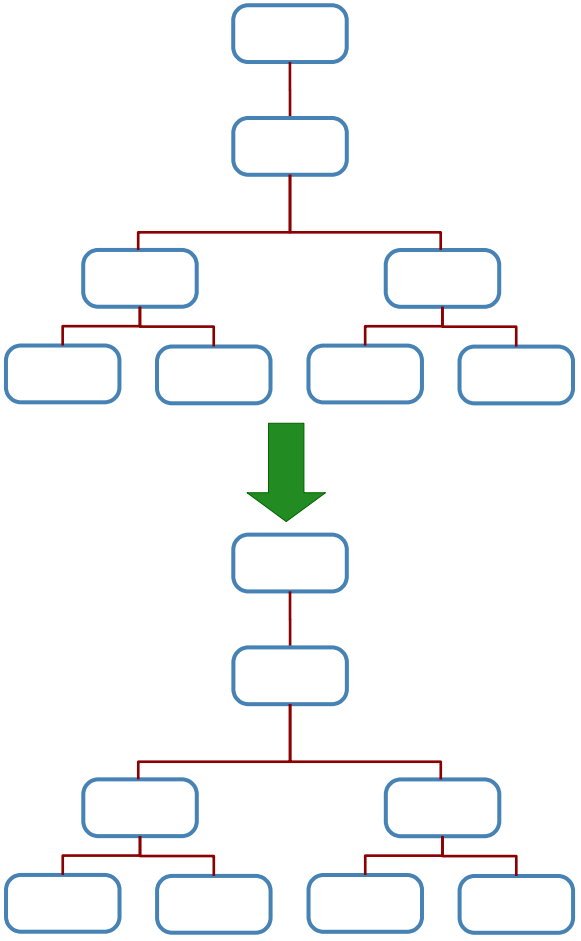


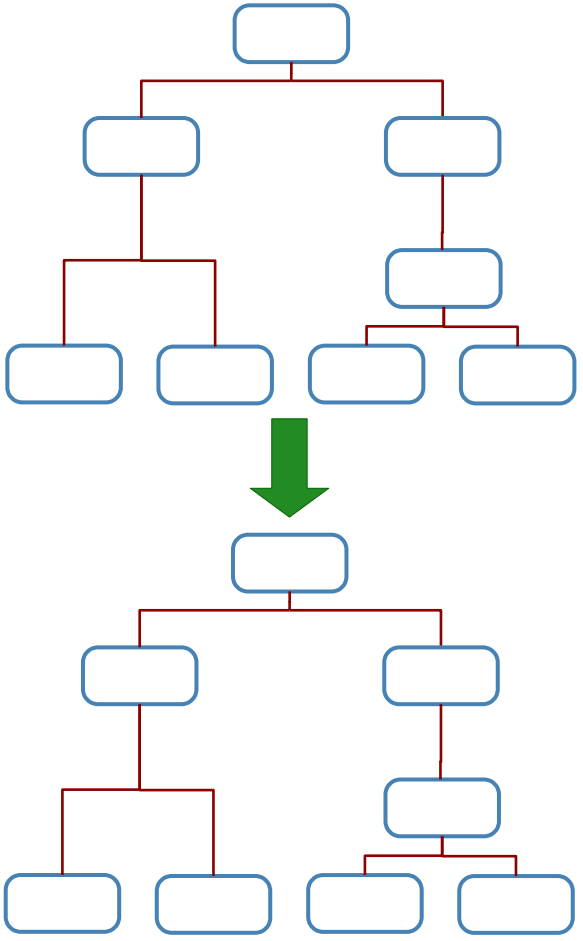


Melvin E. Conway’s Law

How do Committees Invent? in Datamation, vol. 14, num. 4, pp. 28–31, 1968

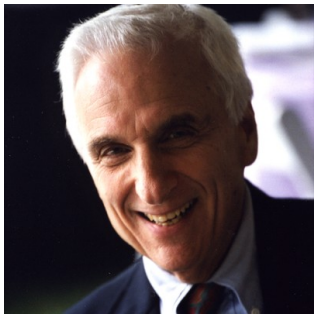


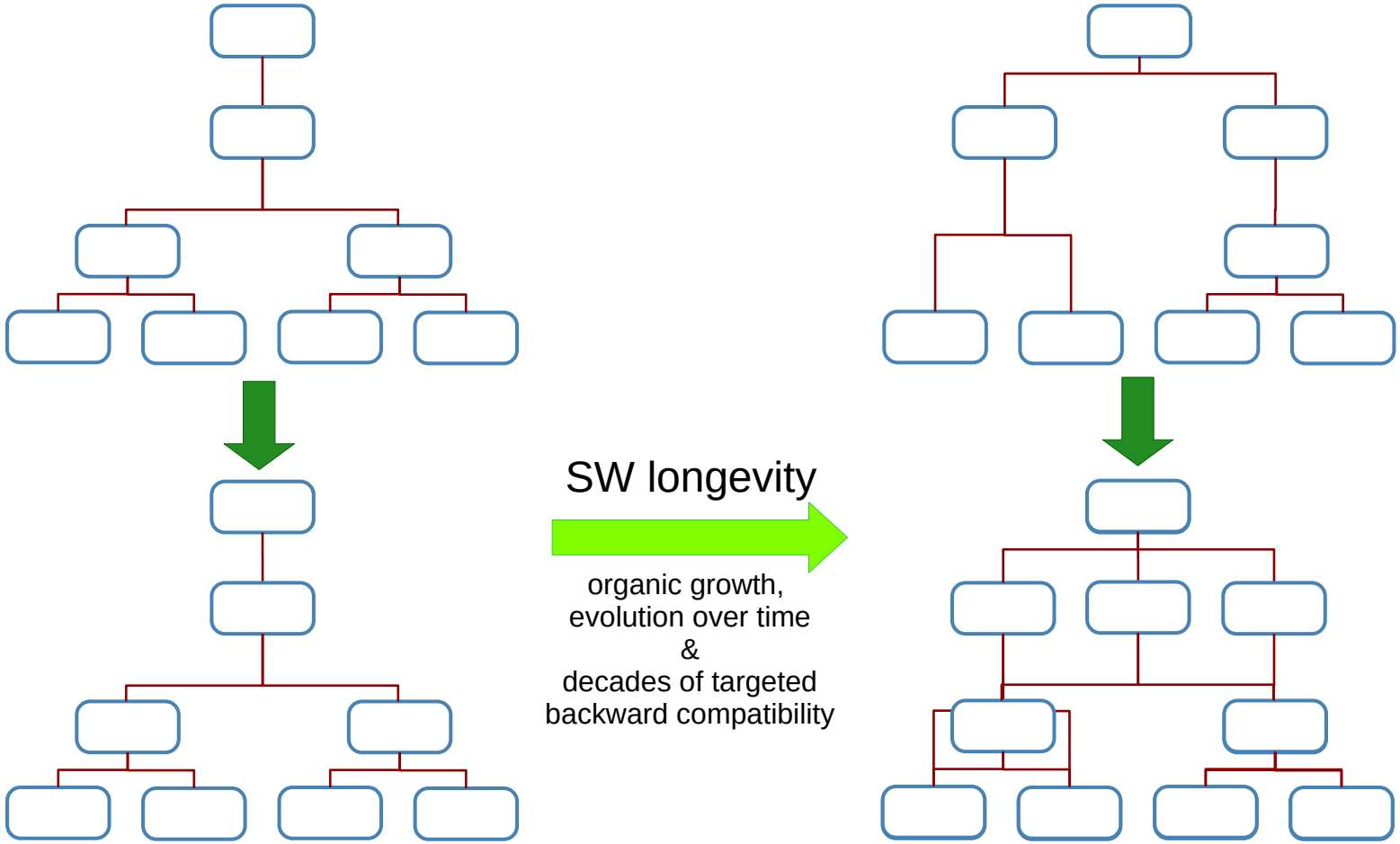




Melvin E. Conway's Law

How do Committees Invent? in Datamation, vol. 14, num. 4, pp. 28–31, 1968

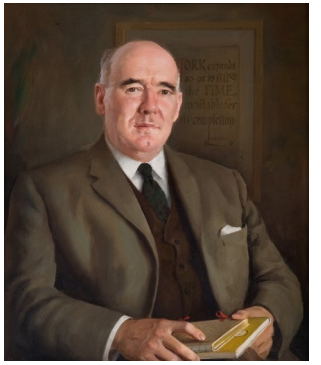




Parkinson’s Law of Triviality – aka. ‘bikeshedding’ effect

Parkinson's Law, or the Pursuit of Progress, John Murray & The Economist, 1958

People within an organization commonly give disproportionate weight to trivial issues.



Cyril Northcote Parkinson
1909 - 1993

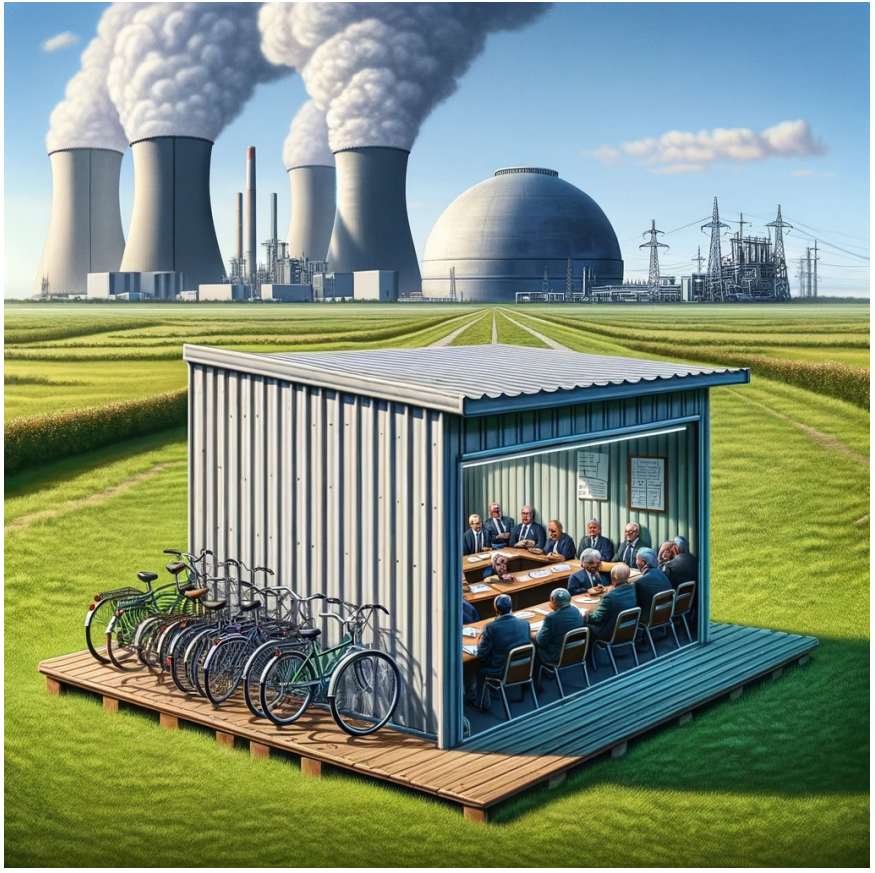
[1] Descamps, A., Massoni, S. & Page, L. Learning to hesitate. Exp Econ 25, 359–383 (2022). https://doi.org/10.1007/s10683-021-09718-7

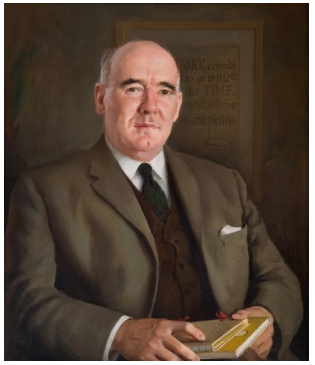
[2] Mikhail Gorbachev: *"Parkinson's law works everywhere." in John O'Sullivan, "Margaret Thatcher: A Legacy of Freedom". Imprimis. Hillsdale College. 37 (6): 6., 2008*

Parkinson’s Law of Triviality – aka. ‘bikeshedding’ effect

[Parkinson's Law, or the Pursuit of Progress](), John Murray & The Economist, 1958

People within an organization commonly give disproportionate weight to trivial issues.





Cyril Northcote Parkinson
1909 - 1993

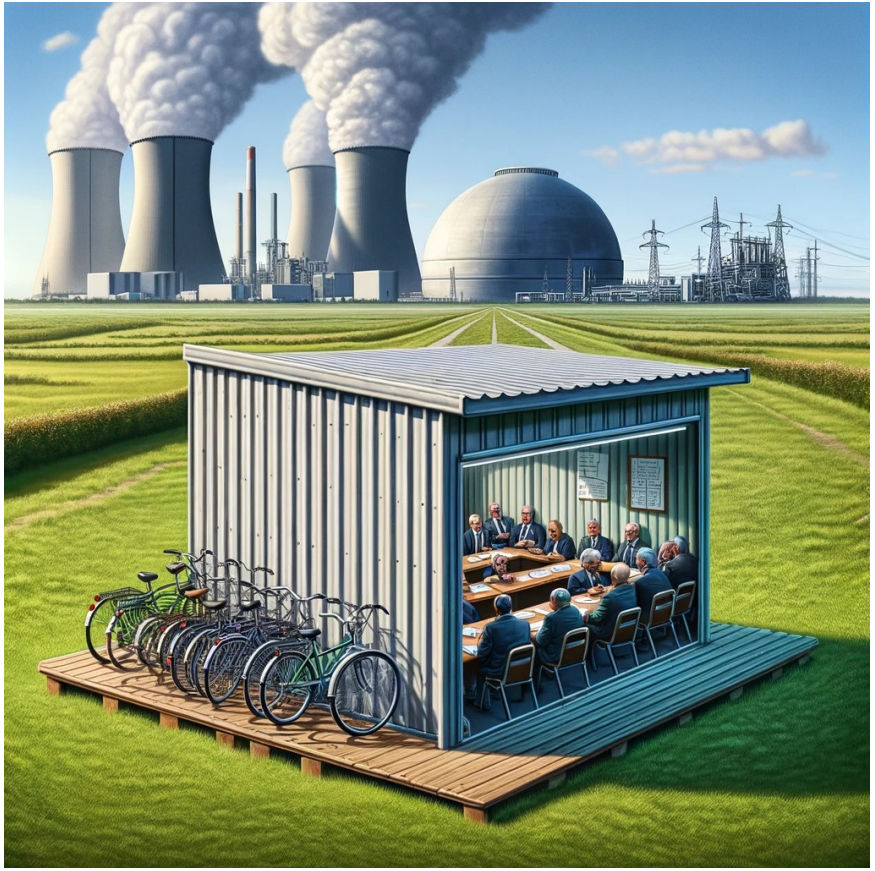
[1] Descamps, A., Massoni, S. & Page, L. Learning to hesitate. Exp Econ 25, 359–383 (2022). https://doi.org/10.1007/s10683-021-09718-7

[2] Mikhail Gorbachev: *"Parkinson's law works everywhere." in John O'Sullivan, "Margaret Thatcher: A Legacy of Freedom". Imprimis. Hillsdale College. 37 (6): 6., 2008*

Parkinson’s Law of Triviality – aka. ‘bikeshedding’ effect

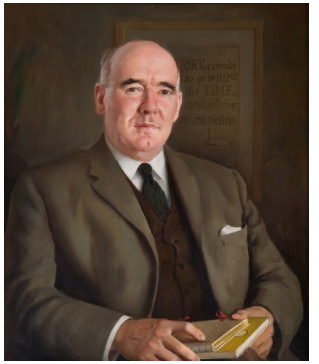
[Parkinson's Law, or the Pursuit of Progress](), John Murray & The Economist, 1958

People within an organization commonly give disproportionate weight to trivial issues.



1. Nuclear Power Plant
   - costs: 28 B€
   - time spent on discussion: 5 minutes

2. Bicycle Shed
   - costs: 10 k€
   - time spent on discussion: 55 minutes



Cyril Northcote Parkinson
1909 - 1993

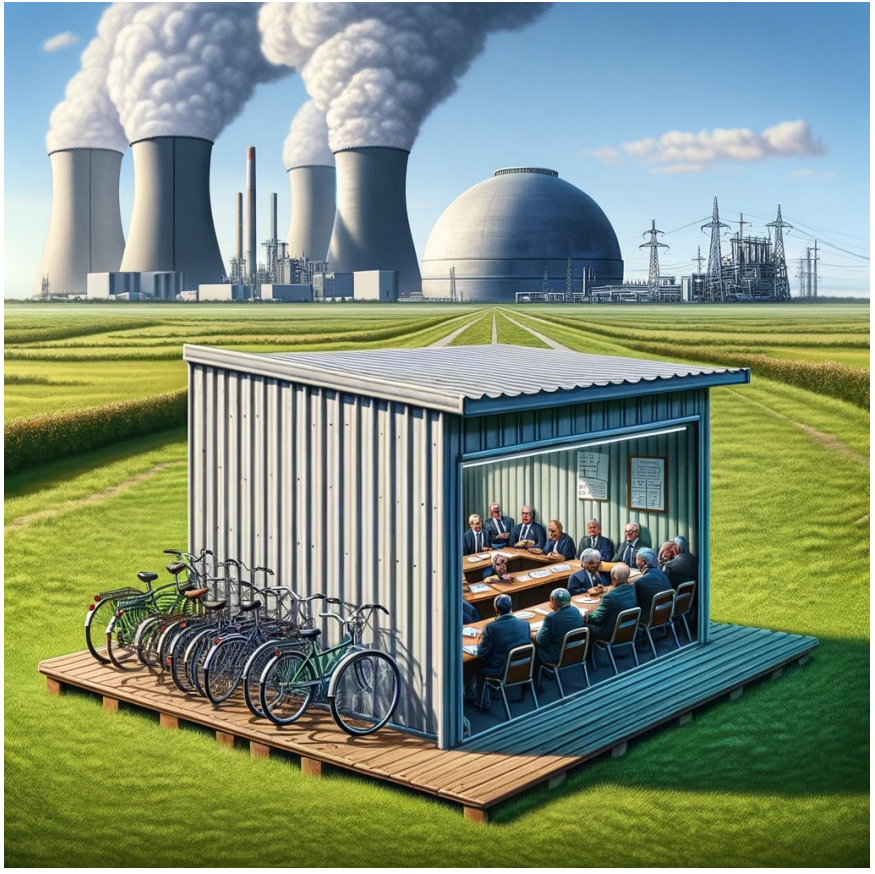
[1] Descamps, A., Massoni, S. & Page, L. Learning to hesitate. Exp Econ 25, 359–383 (2022). https://doi.org/10.1007/s10683-021-09718-7

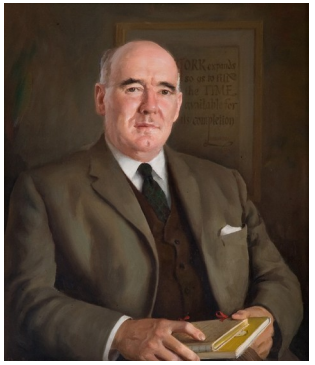
[2] Mikhail Gorbachev: *"Parkinson's law works everywhere." in John O'Sullivan, "Margaret Thatcher: A Legacy of Freedom". Imprimis. Hillsdale College. 37 (6): 6., 2008*

Parkinson’s Law of Triviality – aka. ‘bikeshedding’ effect

Parkinson's Law, or the Pursuit of Progress, John Murray & The Economist, 1958

People within an organization commonly give disproportionate weight to trivial issues.





Cyril Northcote Parkinson
1909 - 1993

1. Nuclear Power Plant
   - costs: 28 B€
   - time spent on discussion: 5 minutes

2. Bicycle Shed
   - costs: 10 k€
   - time spent on discussion: 55 minutes

Possible Remedies:

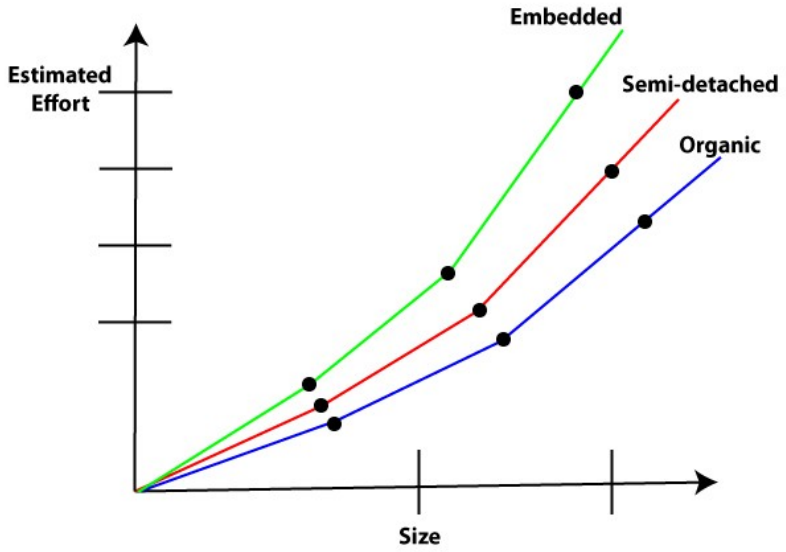
1. be scientific about it: prioritise RSE decisions on observable metrics notably value-for-investment
   - requires set of simple & transparent metrics → SW Guidelines
2. set common standards and metric thresholds
   → automate trivialities through static analyser & linters

[1] Descamps, A., Massoni, S. & Page, L. Learning to hesitate. Exp Econ 25, 359–383 (2022). https://doi.org/10.1007/s10683-021-09718-7

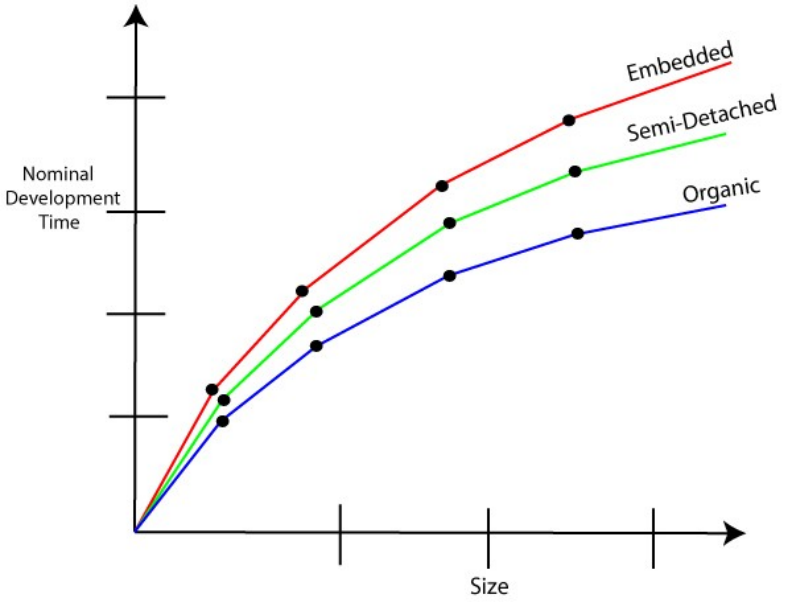
[2] Mikhail Gorbachev: *"Parkinson's law works everywhere." in John O'Sullivan, "Margaret Thatcher: A Legacy of Freedom". Imprimis. Hillsdale College. 37 (6): 6., 2008*

Total Cost of Ownership, Code-Bloat & Source-Lines-Of-Code (SLOCs) I/III

e.g. COCOMO-II Model (N.B. based on 81 + 163 SW projects)



Effort versus product size



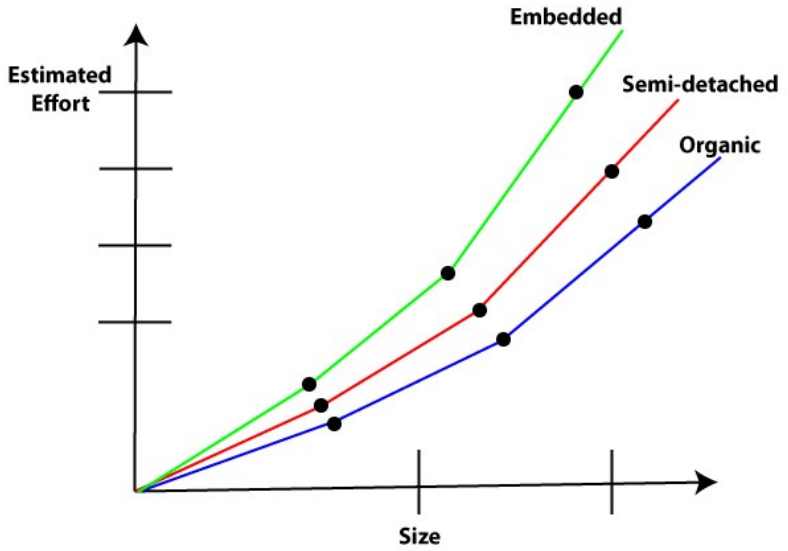
Development time versus size

[1] Boehm, Barry (1981). Software Engineering Economics. Prentice-Hall. ISBN 0-13-822122-7.

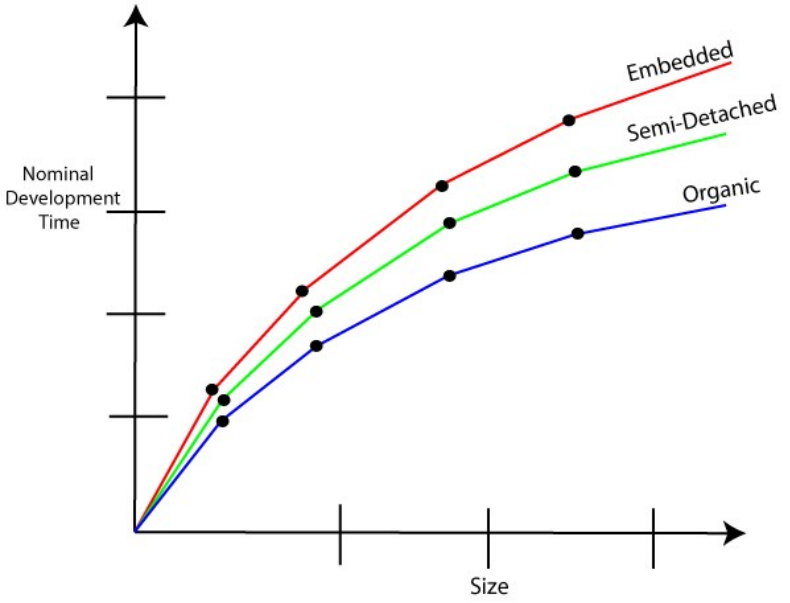
[2] Barry Boehm et al.. Software Cost Estimation with COCOMO II. Englewood Cliffs, NJ:Prentice-Hall, 2000. ISBN 0-13-026692-2

Total Cost of Ownership, Code-Bloat & Source-Lines-Of-Code (SLOCs) I/III

e.g. COCOMO-II Model (N.B. based on 81 + 163 SW projects)



Effort versus product size



Development time versus size

Effort [person-month] $= a_i \cdot (\text{kSLOCs})^{b_i} \cdot \text{Modifier}$

Development Time [month] $= 2.5\, E^{C_i}$

Software project	a_i	b_i	c_i
Organic	3.2	1.05	0.38
Semi-detached	3.0	1.12	0.35
Embedded	2.8	1.20	0.32

[1] Boehm, Barry (1981). Software Engineering Economics. Prentice-Hall. ISBN 0-13-822122-7.

[2] Barry Boehm et al.. Software Cost Estimation with COCOMO II. Englewood Cliffs, NJ:Prentice-Hall, 2000. ISBN 0-13-026692-2

Total Cost of Ownership, Code-Bloat & Source-Lines-Of-Code (SLOCs) II/III

e.g. COCOMO-II Model – EAF Modifier

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database		0.94	1.00	1.08	1.16	
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65
Hardware attributes						
Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
Required turnabout time		0.87	1.00	1.07	1.15	
Personnel attributes						
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

[1] Boehm, Barry (1981). Software Engineering Economics. Prentice-Hall. ISBN 0-13-822122-7.

[2] Barry Boehm et al.. Software Cost Estimation with COCOMO II. Englewood Cliffs, NJ:Prentice-Hall, 2000. ISBN 0-13-026692-2

COCOMO-II Model Example: OpenCMW

```
SLOC    Directory       SLOC-by-Language (Sorted)
26982   src             cpp=26982
1852    concepts        cpp=1852
0       cmake           (none)
0       docs            (none)
0       top_dir         (none)


Totals grouped by language (dominant language first):
cpp:          28834 (100.00%)



Total Physical Source Lines of Code (SLOC)                = 28,834
Development Effort Estimate, Person-Years (Person-Months) = 6.82 (81.87)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 1.11 (13.33)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 6.14
Total Estimated Cost to Develop                           = $ 921,599
 (average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
SLOCCount is Open Source Software/Free Software, licensed under the GNU GPL.
SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to
redistribute it under certain conditions as specified by the GNU GPL license;
see the documentation for details.
Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."
```

FAIR GSI

Total Cost of Ownership, Code-Bloat & Source-Lines-Of-Code (SLOCs)

Why it matters … (ignoring short-lived projects)

- Software must be adaptable to frequent changes and requirements

Total Cost of Ownership, Code-Bloat & Source-Lines-Of-Code (SLOCs)

Why it matters … (ignoring short-lived projects)

- Software must be adaptable to frequent changes and requirements
  → two life-cycle options:
  A) write-once-and-forget → wait for <*N*> years → rinse & repeat

Total Cost of Ownership, Code-Bloat & Source-Lines-Of-Code (SLOCs)

Why it matters … (ignoring short-lived projects)

- Software must be adaptable to frequent changes and requirements
  → two life-cycle options:
  A) write-once-and-forget → wait for <*N*> years → rinse & repeat
  B) write-once & continuously rejuvenate project over <*N*> years

Total Cost of Ownership, Code-Bloat & Source-Lines-Of-Code (SLOCs)

Why it matters … (ignoring short-lived projects)

- Software must be adaptable to frequent changes and requirements
  → two life-cycle options:
  A) write-once-and-forget → wait for <*N*> years → rinse & repeat
  B) write-once & continuously rejuvenate project over <*N*> years

- Things to consider:
  - how to retain software and domain expertise?
    - onboarding of new and retaining of experienced developers? ‘Bus Factor’!

Total Cost of Ownership, Code-Bloat & Source-Lines-Of-Code (SLOCs)

Why it matters … (ignoring short-lived projects)

- Software must be adaptable to frequent changes and requirements
  → two life-cycle options:
  A) write-once-and-forget → wait for <*N*> years → rinse & repeat
  B) write-once & continuously rejuvenate project over <*N*> years

- Things to consider:
  - how to retain software and domain expertise?
    - onboarding of new and retaining of experienced developers? ‘Bus Factor’!
  - maintenance effort profile: spike every <N> years or continuous resource commitment
    - which provides least surprise? highest operational reliability? flexibility? engagement of developer? …
    - ‘starting from scratch’ vs. ‘smooth transitions’
      (i.e. ‘the bandage problem’ vs. German emphasis on minimising risks vs. investing into opportunities)

Total Cost of Ownership, Code-Bloat & Source-Lines-Of-Code (SLOCs)

Why it matters … (ignoring short-lived projects)

- Software must be adaptable to frequent changes and requirements
  → two life-cycle options:
  A) write-once-and-forget → wait for <*N*> years → rinse & repeat
  B) write-once & continuously rejuvenate project over <*N*> years

- Things to consider:
  - how to retain software and domain expertise?
    - onboarding of new and retaining of experienced developers? ‘Bus Factor’!
  - maintenance effort profile: spike every <N> years or continuous resource commitment
    - which provides least surprise? highest operational reliability? flexibility? engagement of developer? …
    - ‘starting from scratch’ vs. ‘smooth transitions’
      (i.e. ‘the bandage problem’ vs. German emphasis on minimising risks vs. investing into opportunities)
  - delayed or forced maintenance (e.g. mitigating IT security risks) may come at an inconvenient time when systems break catastrophically or are compromised.

Total Cost of Ownership, Code-Bloat & Source-Lines-Of-Code (SLOCs)

Why it matters … (ignoring short-lived projects)

- Software must be adaptable to frequent changes and requirements
  → two life-cycle options:
  A) write-once-and-forget → wait for <*N*> years → rinse & repeat
  B) write-once & continuously rejuvenate project over <*N*> years

- Things to consider:
  - how to retain software and domain expertise?
    - onboarding of new and retaining of experienced developers? ‘Bus Factor’!
  - maintenance effort profile: spike every <N> years or continuous resource commitment
    - which provides least surprise? highest operational reliability? flexibility? engagement of developer? …
    - ‘starting from scratch’ vs. ‘smooth transitions’
      (i.e. ‘the bandage problem’ vs. German emphasis on minimising risks vs. investing into opportunities)
  - delayed or forced maintenance (e.g. mitigating IT security risks) may come at an inconvenient time when systems break catastrophically or are compromised.

- typical technology & staff rotation periods ~ 3-5 years
  ↔ total cost-of-ownership = ‘initial kSLOCs’ + ‘initial kSLOCs’/~5yrs

FAIR GSI

Code-Bloat, Source-Lines-Of-Code (SLOCs), and Total Cost of Ownership

Why it matters … (ignoring short-lived projects) contd.

- typical technology & staff rotation periods ~ 3-5 years
  ↔ total cost-of-ownership = ‘initial kSLOCs’ + ‘initial kSLOCs’/~5yrs

Code-Bloat, Source-Lines-Of-Code (SLOCs), and Total Cost of Ownership

Why it matters … (ignoring short-lived projects) contd.

- typical technology & staff rotation periods ~ 3-5 years
  ↔ total cost-of-ownership = ‘initial kSLOCs’ + ‘initial kSLOCs’/~5yrs

Code-Bloat, Source-Lines-Of-Code (SLOCs), and Total Cost of Ownership

Why it matters … (ignoring short-lived projects) contd.

- typical technology & staff rotation periods ~ 3-5 years
  ↔ total cost-of-ownership = ‘initial kSLOCs’ + ‘initial kSLOCs’/~5yrs

- disclaimer: these indicators and metrics not not absolute unconditional truths
  – nevertheless the less SLOCs:

Code-Bloat, Source-Lines-Of-Code (SLOCs), and Total Cost of Ownership

Why it matters … (ignoring short-lived projects) contd.

- typical technology & staff rotation periods ~ 3-5 years
  ↔ total cost-of-ownership = ‘initial kSLOCs’ + ‘initial kSLOCs’/~5yrs

- disclaimer: these indicators and metrics not not absolute unconditional truths – nevertheless the less SLOCs:
  - … the less code to read and the easier to onboard new developers
  - … the easier to reason about logic and intent

Code-Bloat, Source-Lines-Of-Code (SLOCs), and Total Cost of Ownership

Why it matters … (ignoring short-lived projects) contd.

- typical technology & staff rotation periods ~ 3-5 years
  ↔ total cost-of-ownership = ‘initial kSLOCs’ + ‘initial kSLOCs’/~5yrs

- disclaimer: these indicators and metrics not not absolute unconditional truths
  – nevertheless the less SLOCs:
  - … the less code to read and the easier to onboard new developers
  - … the easier to reason about logic and intent
  - … the easier to re-engineer the same functionality
    with a new, optimised and possibly easier to maintain implementation

Code-Bloat, Source-Lines-Of-Code (SLOCs), and Total Cost of Ownership

Why it matters … (ignoring short-lived projects) contd.

- typical technology & staff rotation periods ~ 3-5 years
  ↔ total cost-of-ownership = ‘initial kSLOCs’ + ‘initial kSLOCs’/~5yrs

- disclaimer: these indicators and metrics not not absolute unconditional truths
  – nevertheless the less SLOCs:
  - … the less code to read and the easier to onboard new developers
  - … the easier to reason about logic and intent
  - … the easier to re-engineer the same functionality
    with a new, optimised and possibly easier to maintain implementation
  - … the easier to debug, to unit-/integration-test, and thus fewer bugs
  - … the easier to achieve compliance with safety and security standards
  - … the easier to optimise for computational and memory performance
  - … easier collaboration and code-reuse

HowTo (realistically) a) assess and b) mitigate Risks related to RSE

Assess & Identify:

- potential vendor lock-in
- maintenance risks (code-bloat, …)
- safety & security (human, IT data, machine)
- Unnecessary feature or capability duplication (aka. ‘reinventing the wheel’)
- CI/CD → level of unit- and integration tests

→ needs agreed-upon metrics

→ much easier if FLOSS:

- in-house and external peer-review
- professional grade static-code analysis (e.g. Sonar, Snyk, Coverity, …)
- significantly lowers costs of onboarding
  - new staff, users, exp. collaborators, ...
  - industry (also for maintenance contracts)

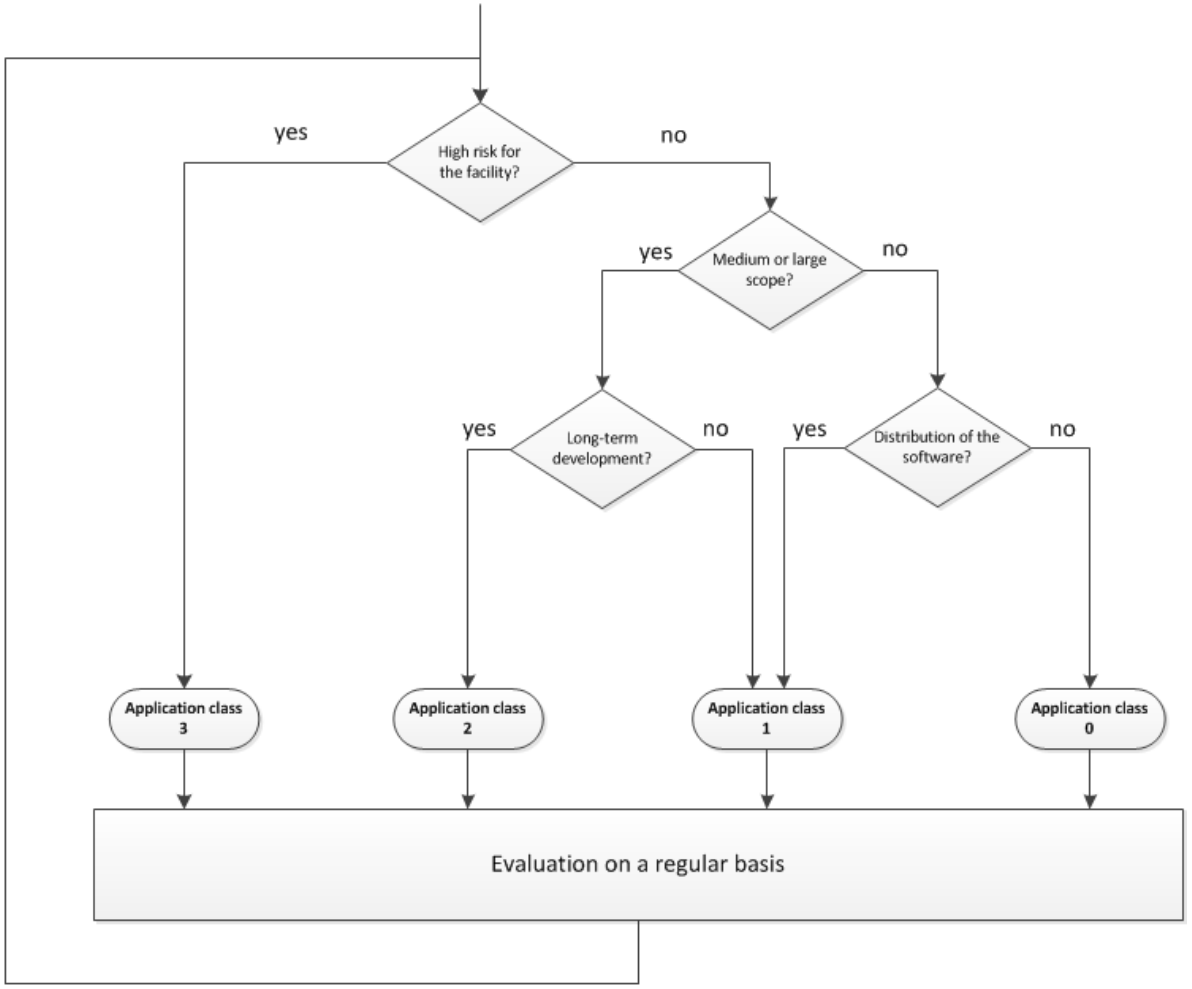
Train and Develop Software Engineers:

- Best Practices:
  - C++ User-Group
  - modern development standards
  - … *(your suggestions)*
- Entice, foster and enforce code quality
  → simplify onboarding & in-house/ext collaborations
  - minimal CI/CD & reporting standards
  - Core Development, Naming & Formatting Guidelines
  - … *(your suggestions)*
- Ethics – not just ‘Code of Conduct’
  - Positive Community Engagement
    “be an ambassador of FAIR”, aids:
    - intellectual cross-pollination & enticing external contributions,
    - recruitment, procurement, …
    - organisational reputation, ...
    - impact on society → funding!

FAIR GSI

HowTo (realistically) a) assess and b) mitigate Risks related to RSE

Application Classification Example DLR



HowTo (realistically) a) assess and b) mitigate Risks related to RSE

Application Classification Proposal for GSI/FAIR

HowTo (realistically) a) assess and b) mitigate Risks related to RSE

Application Classification Proposal for GSI/FAIR

Application Class 0
(1 dev & ≤ 5 kEUR)

Decision: dev ↔ group/dep. lead

greater scrutiny with greater risks

employee

HowTo (realistically) a) assess and b) mitigate Risks related to RSE

Application Classification Proposal for GSI/FAIR

Application Class 0
(1 dev & ≤ 5 kEUR)
Decision: dev ↔ group/dep. lead

Examples:

- < 2 kSLOCs analysis scripts
- spread-sheet macros
- …
- reproducible within days

greater scrutiny with greater risks

employee

HowTo (realistically) a) assess and b) mitigate Risks related to RSE

Application Classification Proposal for GSI/FAIR

Application Class 1
(< 8 devs & ≤ 30 kEUR)

Decision: group/dep. ↔ div lead.

Application Class 0
(1 dev & ≤ 5 kEUR)

Decision: dev ↔ group/dep. lead

Examples:

- < 2 kSLOCs analysis scripts
- spread-sheet macros
- …
- reproducible within days

greater scrutiny with greater risks

employee

HowTo (realistically) a) assess and b) mitigate Risks related to RSE

Application Classification Proposal for GSI/FAIR

Application Class 0
(1 dev & ≤ 5 kEUR)
Decision: dev ↔ group/dep. lead

Examples:

- < 2 kSLOCs analysis scripts
- spread-sheet macros
- …
- reproducible within days

Application Class 1
(< 8 devs & ≤ 30 kEUR)
Decision: group/dep. ↔ div lead.

Examples:

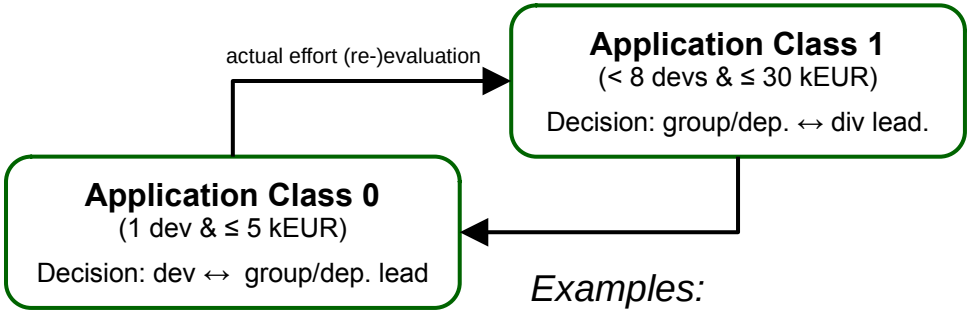
- < 100 kSLOCs libraries/tools
- …
- reproducible within few engineering months

greater scrutiny with greater risks

employee

HowTo (realistically) a) assess and b) mitigate Risks related to RSE

Application Classification Proposal for GSI/FAIR



Examples:

- < 2 kSLOCs analysis scripts
- spread-sheet macros
- …
- reproducible within days

Examples:

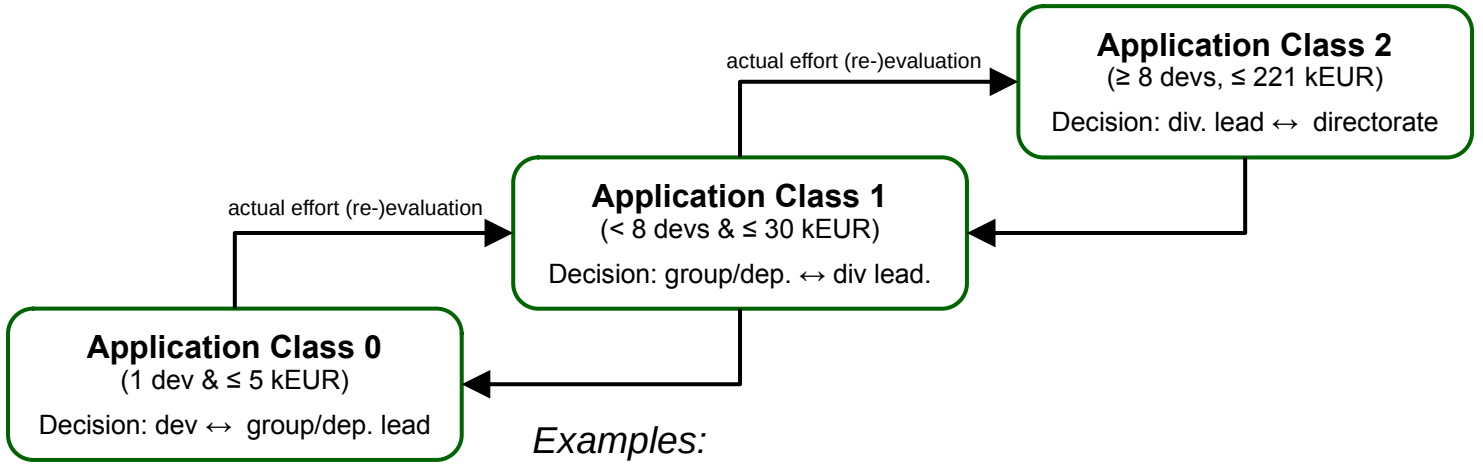
- < 100 kSLOCs libraries/tools
- …
- reproducible within few engineering months

greater scrutiny with greater risks

employee

HowTo (realistically) a) assess and b) mitigate Risks related to RSE

Application Classification Proposal for GSI/FAIR



Examples:

- < 2 kSLOCs analysis scripts
- spread-sheet macros
- …
- reproducible within days

Examples:

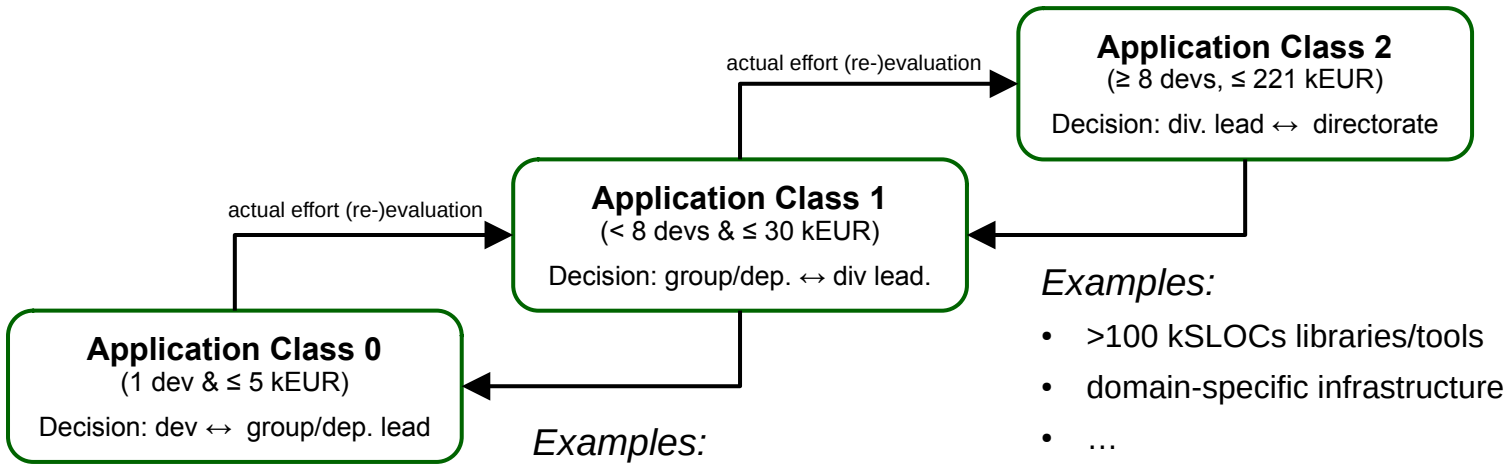
- < 100 kSLOCs libraries/tools
- …
- reproducible within few engineering months

greater scrutiny with greater risks

employee

HowTo (realistically) a) assess and b) mitigate Risks related to RSE

Application Classification Proposal for GSI/FAIR



Examples:

- < 2 kSLOCs analysis scripts
- spread-sheet macros
- …
- reproducible within days

Examples:

- < 100 kSLOCs libraries/tools
- …
- reproducible within few engineering months

Examples:

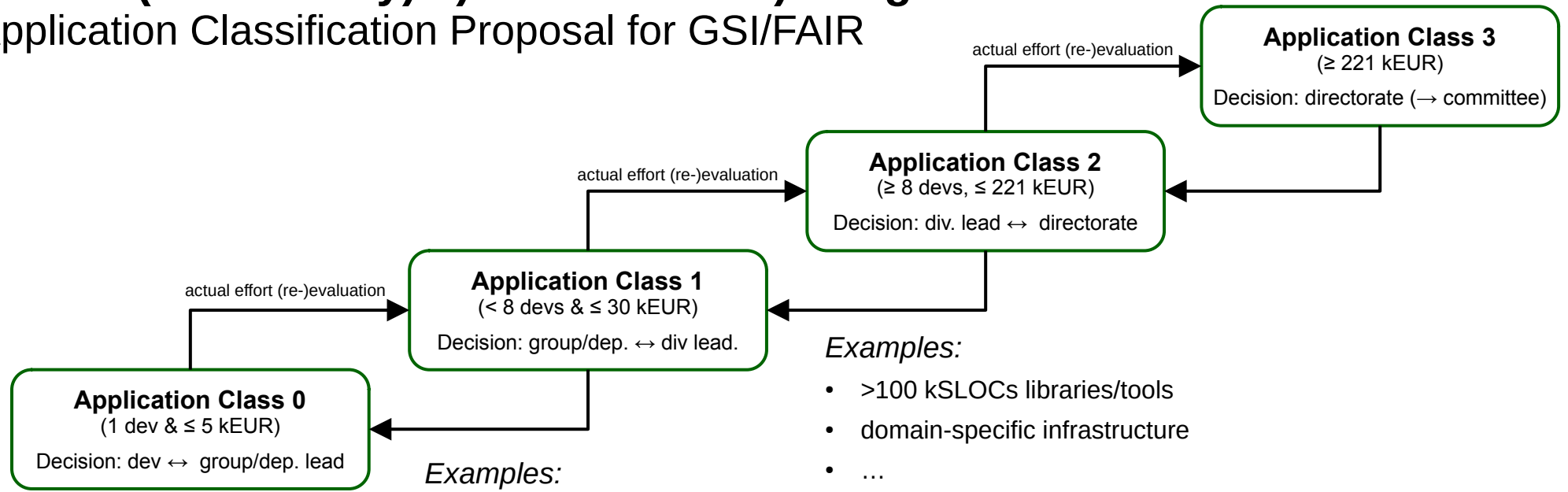
- >100 kSLOCs libraries/tools
- domain-specific infrastructure
- …
- reproducible within few FTEs

greater scrutiny with greater risks

employee

HowTo (realistically) a) assess and b) mitigate Risks related to RSE

Application Classification Proposal for GSI/FAIR



Examples:

- < 2 kSLOCs analysis scripts
- spread-sheet macros
- …
- reproducible within days

Examples:

- < 100 kSLOCs libraries/tools
- …
- reproducible within few engineering months

Examples:

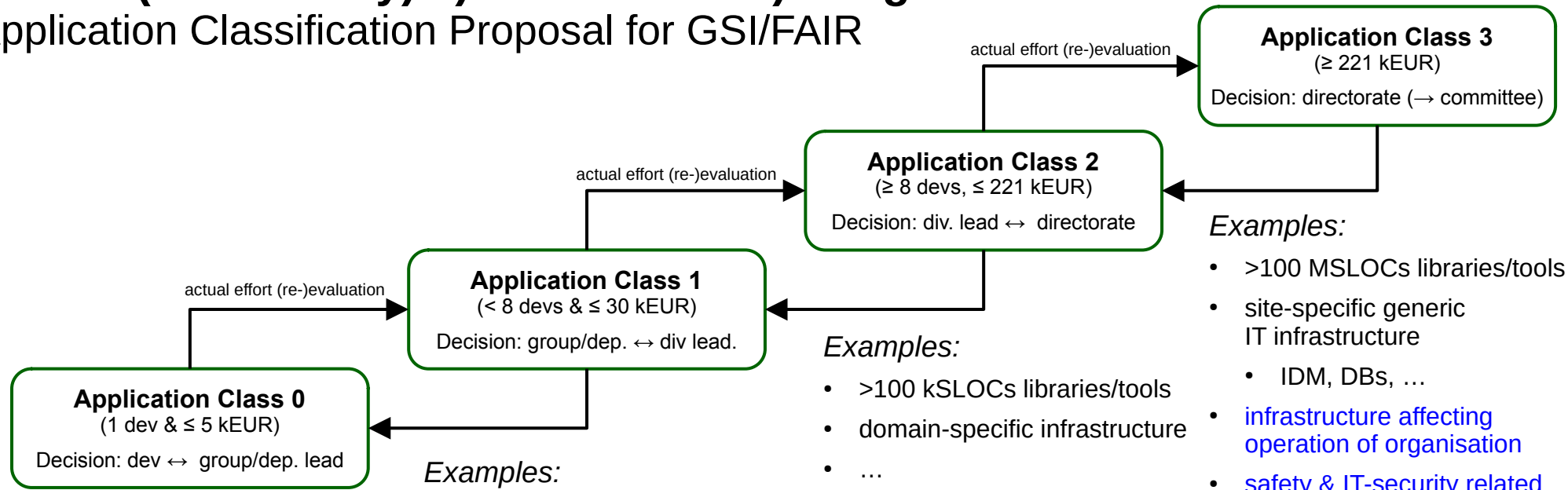
- >100 kSLOCs libraries/tools
- domain-specific infrastructure
- …
- reproducible within few FTEs

greater scrutiny with greater risks

employee

HowTo (realistically) a) assess and b) mitigate Risks related to RSE

Application Classification Proposal for GSI/FAIR



Examples:

- < 2 kSLOCs analysis scripts
- spread-sheet macros
- …
- reproducible within days

Examples:

- < 100 kSLOCs libraries/tools
- …
- reproducible within few engineering months

Examples:

- >100 kSLOCs libraries/tools
- domain-specific infrastructure
- …
- reproducible within few FTEs

Examples:

- >100 MSLOCs libraries/tools
- site-specific generic IT infrastructure
  - IDM, DBs, …
- infrastructure affecting operation of organisation
- safety & IT-security related
- non-open-source
- …

greater scrutiny with greater risks

employee

Central Questions regarding Software Guidelines

Central Questions regarding Software Guidelines

- **What constitutes and how to optimise for 'lean' and 'clean' code? Which aspects should be:**
  - **enticed** – Ethics, Netiquette, engaging with collaborators, users, and society, ...
  - **recommended** – GSI/FAIR team standard, simplifying onboarding, cross-departmental support, or
  - **must be enforced** ↔ mitigating costs & risk for the organisation/divisions

Central Questions regarding Software Guidelines

- **What constitutes and how to optimise for 'lean' and 'clean' code? Which aspects should be:**
  - **enticed** – Ethics, Netiquette, engaging with collaborators, users, and society, ...
  - **recommended** – GSI/FAIR team standard, simplifying onboarding, cross-departmental support, or
  - **must be enforced** ↔ mitigating costs & risk for the organisation/divisions

- *'clean code'* and *'lean management'* have sometimes devolved into buzzwords, their origins are rooted in
  - **quantifiable metrics**: software must commit to quantifiable and actionable optimisation standards, beyond 'best practice' incantations.
  - **lean code**: originating from the Toyota Production System (Wikipedia), this philosophy focuses on streamlining processes, eliminating waste (*muda*), and enhancing value creation for both the organization and its users.

    clean code: the principles laid out by Robert C. Martin, encapsulating best practices in programming for clarity and maintainability

At its core: you cannot have one without the other. establishing measurable *'lean'* and *'clean'* coding practices is essential for effective communication and organization within diverse teams.

Central Questions regarding Software Guidelines

- **What constitutes and how to optimise for 'lean' and 'clean' code? Which aspects should be:**
  - **enticed** – Ethics, Netiquette, engaging with collaborators, users, and society, ...
  - **recommended** – GSI/FAIR team standard, simplifying onboarding, cross-departmental support, or
  - **must be enforced** ↔ mitigating costs & risk for the organisation/divisions

- *'clean code'* and *'lean management'* have sometimes devolved into buzzwords, their origins are rooted in
  - **quantifiable metrics**: software must commit to quantifiable and actionable optimisation standards, beyond 'best practice' incantations.
  - **lean code**: originating from the Toyota Production System (Wikipedia), this philosophy focuses on streamlining processes, eliminating waste (*muda*), and enhancing value creation for both the organization and its users.

    clean code: the principles laid out by Robert C. Martin, encapsulating best practices in programming for clarity and maintainability

  At its core: you cannot have one without the other. establishing measurable *'lean'* and *'clean'* coding practices is essential for effective communication and organization within diverse teams.

- Software Guideline's goal: establish principles and standards benefiting individual developers, teams, and management, ultimately improving the outcome and risks for all involved.

Purposeful and Metric-Driven Coding: The 'W.A.R.M.E.D' Approach

Concept courtesy Casey Muratori – for details see here

W.A.R.M.E.D. is a mnemonic inspired by Casey Muratori, advocating for robust, efficient coding without over-engineering. Each principle is tied to a specific metric for optimal success:

Purposeful and Metric-Driven Coding: The 'W.A.R.M.E.D' Approach

Concept courtesy Casey Muratori – for details see here

W.A.R.M.E.D. is a mnemonic inspired by Casey Muratori, advocating for robust, efficient coding without over-engineering. Each principle is tied to a specific metric for optimal success:

1. **Write**: Address real-world problems with simplicity. Use Story Points (SP) to estimate and track effort. Metric: SP (Keep < 5 SP).

Purposeful and Metric-Driven Coding: The 'W.A.R.M.E.D' Approach

Concept courtesy Casey Muratori – for details see here

W.A.R.M.E.D. is a mnemonic inspired by Casey Muratori, advocating for robust, efficient coding without over-engineering. Each principle is tied to a specific metric for optimal success:

1. **Write**: Address real-world problems with simplicity. Use Story Points (SP) to estimate and track effort. Metric: SP (Keep < 5 SP).
2. **Agree**: Choose empirically backed designs over speculative ones. Focus on substantial challenges, avoiding bike-shedding. Metric: Time/SPs in design discussions.

Purposeful and Metric-Driven Coding: The 'W.A.R.M.E.D' Approach

Concept courtesy Casey Muratori – for details see here

W.A.R.M.E.D. is a mnemonic inspired by Casey Muratori, advocating for robust, efficient coding without over-engineering. Each principle is tied to a specific metric for optimal success:

1. **Write**: Address real-world problems with simplicity. Use Story Points (SP) to estimate and track effort. Metric: SP (Keep < 5 SP).
2. **Agree**: Choose empirically backed designs over speculative ones. Focus on substantial challenges, avoiding bike-shedding. Metric: Time/SPs in design discussions.
3. **Readable**: Code as clear prose. Reduce SLOCs, using COCOMO for simplicity. Metric: SLOCs.

Purposeful and Metric-Driven Coding: The 'W.A.R.M.E.D' Approach

Concept courtesy Casey Muratori – for details see here

W.A.R.M.E.D. is a mnemonic inspired by Casey Muratori, advocating for robust, efficient coding without over-engineering. Each principle is tied to a specific metric for optimal success:

1. **Write**: Address real-world problems with simplicity. Use Story Points (SP) to estimate and track effort. Metric: SP (Keep < 5 SP).
2. **Agree**: Choose empirically backed designs over speculative ones. Focus on substantial challenges, avoiding bike-shedding. Metric: Time/SPs in design discussions.
3. **Readable**: Code as clear prose. Reduce SLOCs, using COCOMO for simplicity. Metric: SLOCs.
4. **Modify**: Add features/APIs based on clear needs, not hypotheticals. Test software's composability. Metric: SP.

Purposeful and Metric-Driven Coding: The 'W.A.R.M.E.D' Approach

Concept courtesy Casey Muratori – for details see here

W.A.R.M.E.D. is a mnemonic inspired by Casey Muratori, advocating for robust, efficient coding without over-engineering. Each principle is tied to a specific metric for optimal success:

1. **Write**: Address real-world problems with simplicity. Use Story Points (SP) to estimate and track effort. Metric: SP (Keep < 5 SP).
2. **Agree**: Choose empirically backed designs over speculative ones. Focus on substantial challenges, avoiding bike-shedding. Metric: Time/SPs in design discussions.
3. **Readable**: Code as clear prose. Reduce SLOCs, using COCOMO for simplicity. Metric: SLOCs.
4. **Modify**: Add features/APIs based on clear needs, not hypotheticals. Test software's composability. Metric: SP.
5. **Execute with Efficiency**: Prioritize performance and reliability, backed by benchmarks and testing. Metric: Specific to application (e.g., throughput, latency).

Purposeful and Metric-Driven Coding: The 'W.A.R.M.E.D' Approach

Concept courtesy Casey Muratori – for details see here

W.A.R.M.E.D. is a mnemonic inspired by Casey Muratori, advocating for robust, efficient coding without over-engineering. Each principle is tied to a specific metric for optimal success:

1. **Write**: Address real-world problems with simplicity. Use Story Points (SP) to estimate and track effort. Metric: SP (Keep < 5 SP).
2. **Agree**: Choose empirically backed designs over speculative ones. Focus on substantial challenges, avoiding bike-shedding. Metric: Time/SPs in design discussions.
3. **Readable**: Code as clear prose. Reduce SLOCs, using COCOMO for simplicity. Metric: SLOCs.
4. **Modify**: Add features/APIs based on clear needs, not hypotheticals. Test software's composability. Metric: SP.
5. **Execute with Efficiency**: Prioritize performance and reliability, backed by benchmarks and testing. Metric: Specific to application (e.g., throughput, latency).
6. **Debug Strategically & Use Tools**: Focus on macro-level optimizations. Utilize static code analysis, AI tools, and compilers. Metric: Beyond SPs (e.g., test coverage percentages).

Purposeful and Metric-Driven Coding: The 'W.A.R.M.E.D' Approach

Concept courtesy Casey Muratori – for details see here

W.A.R.M.E.D. is a mnemonic inspired by Casey Muratori, advocating for robust, efficient coding without over-engineering. Each principle is tied to a specific metric for optimal success:

1. **Write**: Address real-world problems with simplicity. Use Story Points (SP) to estimate and track effort. Metric: SP (Keep < 5 SP).
2. **Agree**: Choose empirically backed designs over speculative ones. Focus on substantial challenges, avoiding bike-shedding. Metric: Time/SPs in design discussions.
3. **Readable**: Code as clear prose. Reduce SLOCs, using COCOMO for simplicity. Metric: SLOCs.
4. **Modify**: Add features/APIs based on clear needs, not hypotheticals. Test software's composability. Metric: SP.
5. **Execute with Efficiency**: Prioritize performance and reliability, backed by benchmarks and testing. Metric: Specific to application (e.g., throughput, latency).
6. **Debug Strategically & Use Tools**: Focus on macro-level optimizations. Utilize static code analysis, AI tools, and compilers. Metric: Beyond SPs (e.g., test coverage percentages).

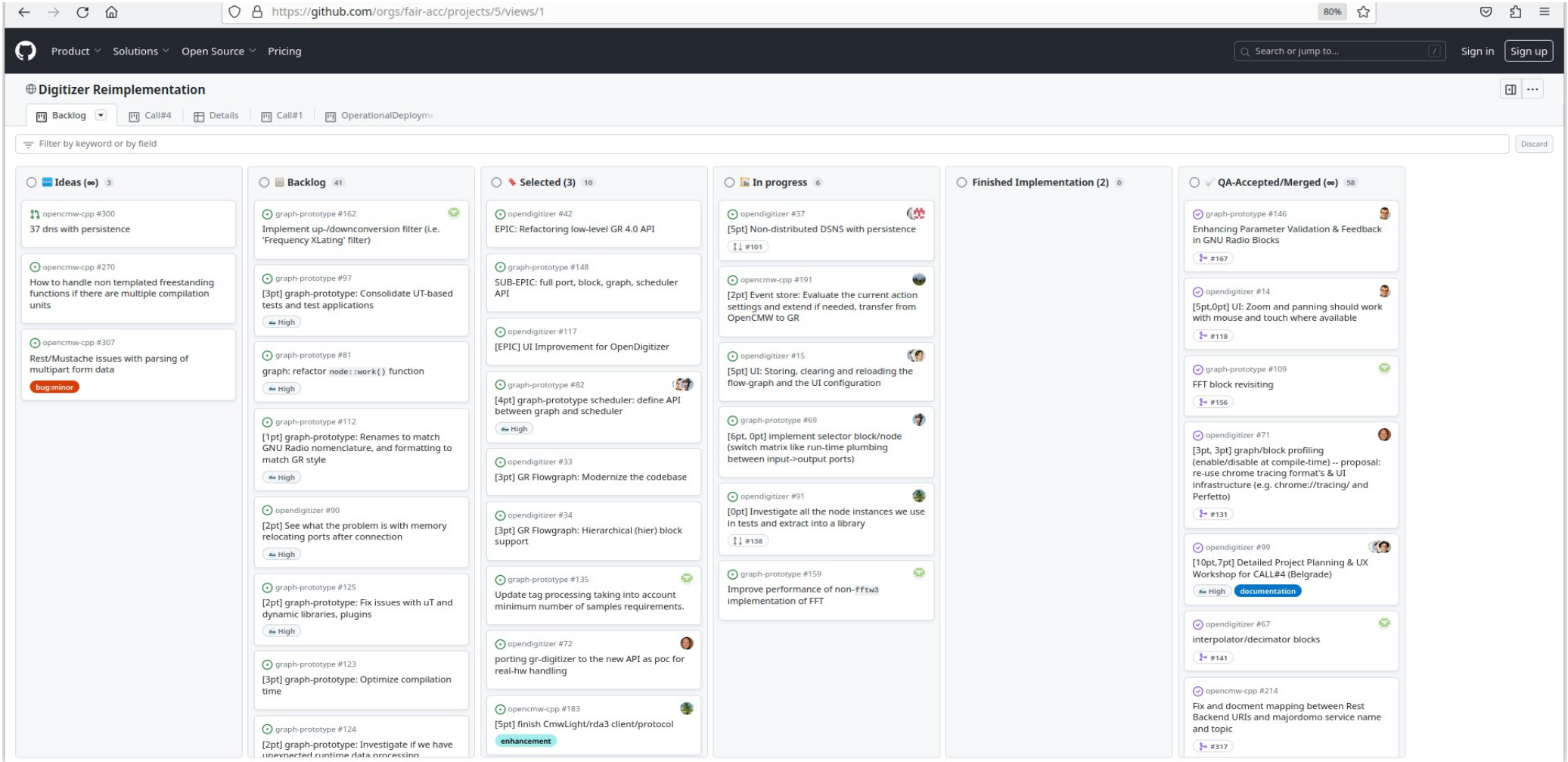
Key Principle: Avoid feature creep and premature optimizations. Apply the 80/20 rule for resource investment, targeting areas with the highest ROI for continuous improvement i.e. ‘biggest bang for the buck’.

Additional Guideline Components:

- Core Naming Conventions – ‘nomen est omen’? e.g. this
- Core Development Guidelines? e.g. this
  - favour focus on C++ Standard → Core- → common in-house Guidelines
  - Specific Development Guidelines, e.g. https://github.com/fair-acc/graph-prototype/blob/main/CORE_DEVELOPMENT_GUIDELINE.md
- Deprecating features of older C++ standards (i.e. C++98/11/14) that are superseded by newer standards w.r.t. safety?
  - pro: ‘spring cleaning’, getting to know your code-base, leaner- & cleaner code
  - con (pro?): lack of development resources becomes more transparent
- […]
- Minimum CI/CD standards? e.g.
  - minimum compiler warning levels? -Werror? Sanitizers?
  - penalising unsafe practices (e.g. C++98/11) superseded by the present C++ standard?
  - Static Code Analysis Tools (Sonar, Snyk, Coverity, ...)

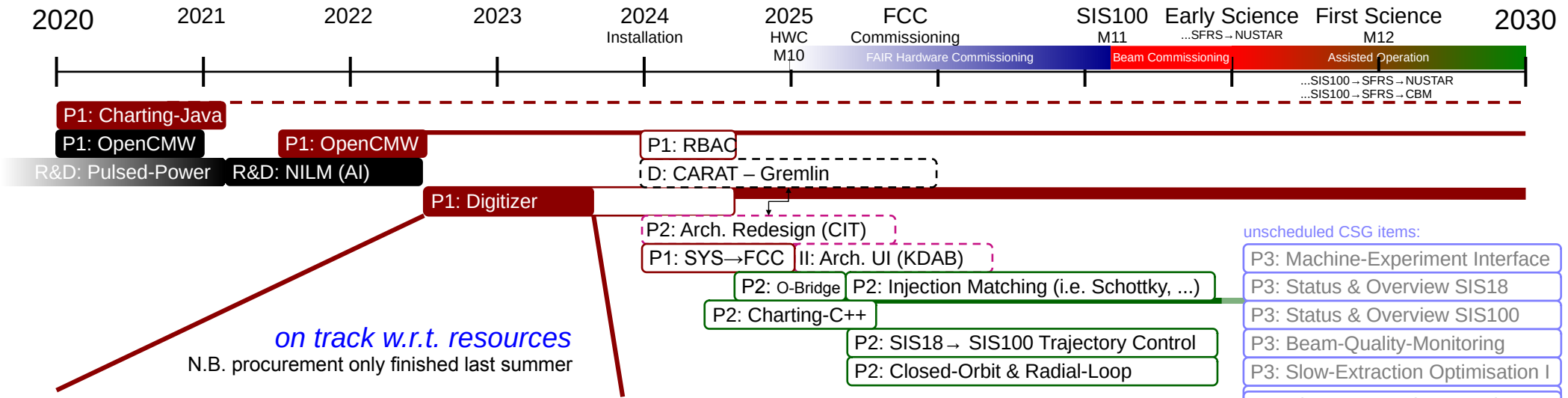
RSE Project Reporting – Kanban Project Board View

specific example: https://github.com/orgs/fair-acc/projects/5/views/1



RSE Project Reporting – Kanban Project Board Derived Metrics

specific example: our PSP 2.14.17 (our FAIR Acc. Sub-Project)

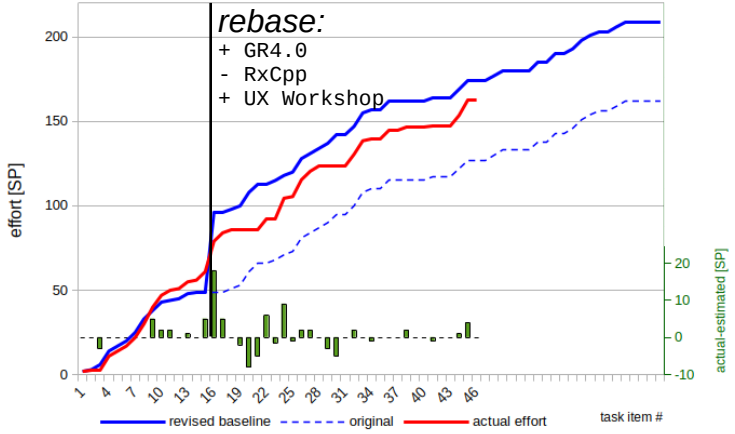


on track w.r.t. resources

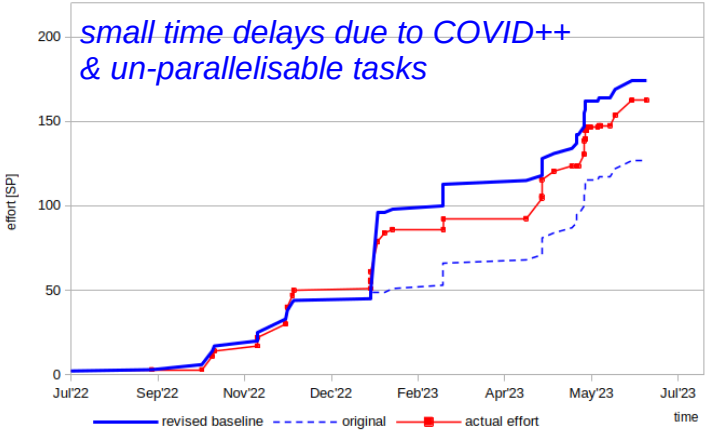
N.B. procurement only finished last summer

unscheduled CSG items:

- P3: Machine-Experiment Interface
- P3: Status & Overview SIS18
- P3: Status & Overview SIS100
- P3: Beam-Quality-Monitoring
- P3: Slow-Extraction Optimisation I
- P3: Dig. Set-Actual Comparison
- P4: el. Pulsed-Load Monitoring
- P4: web-status/fixed-displays
- P4: multi-parameter-beam
- P4: Q/Q' control
- P4: Multi-Turn-Injection Control
- P4: Slow-Extraction Optimisation II
- P4: Optics Correction
- P4: Collimator/Cleaning Set-Up
- P4: Final-Focus Control
- P4: L/T Emittance Control
- P4: Post-Mortem Analysis
- P4: …



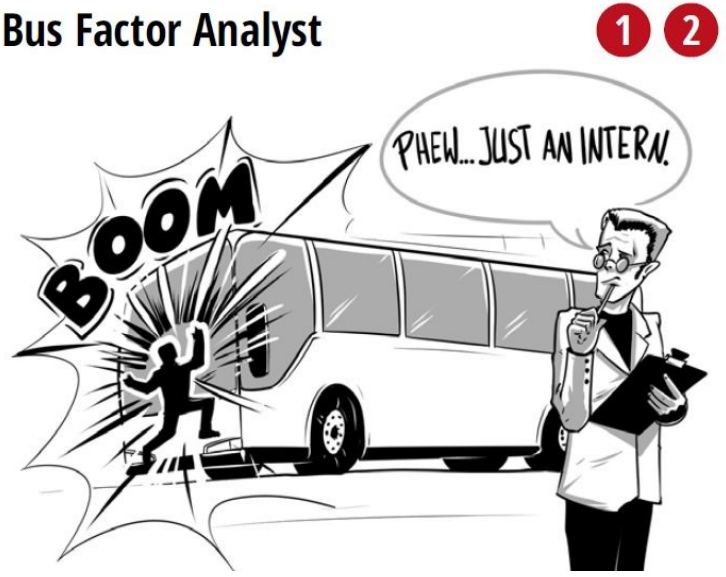
https://github.com/orgs/fair-acc/projects



https://github.com/orgs/fair-acc/projects/5

Metric: Bus Factor & Reusability

… *“number of team members that have to disappear from a project before the project stalls due to lack of knowledgeable or competent personnel"*, Wikipedia



https://www.playitstartup.com/

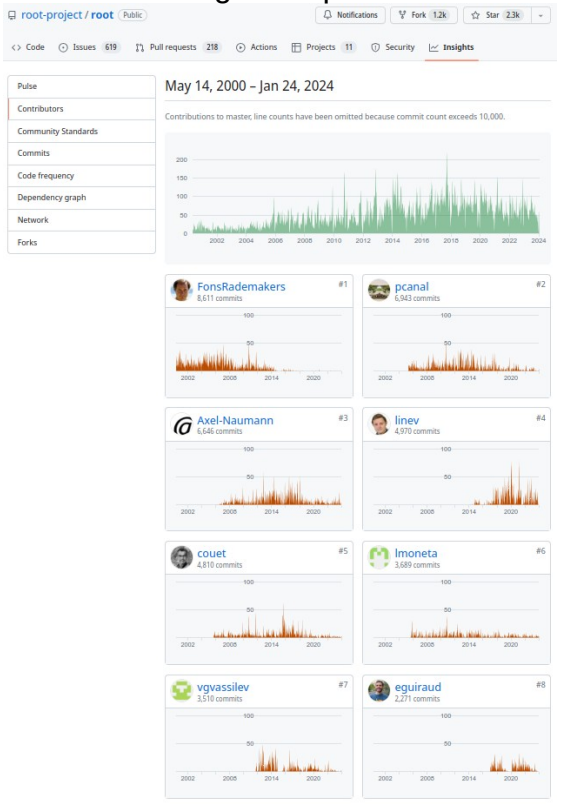
Goal: bus factor ≥ 3

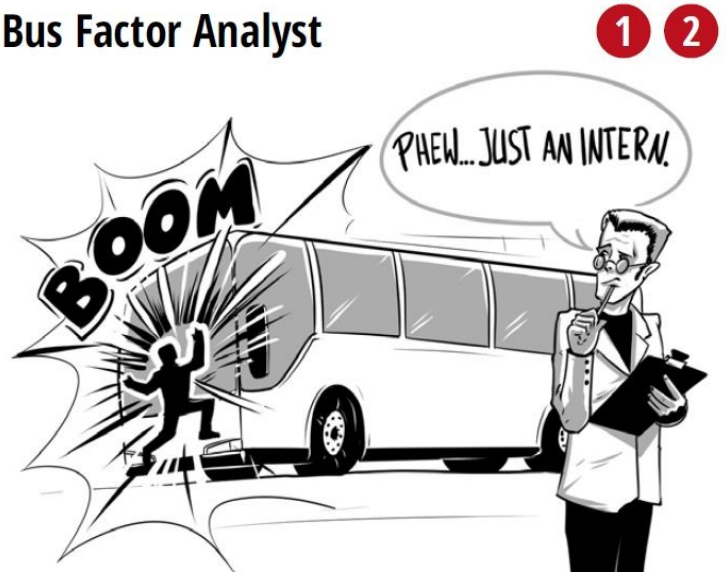
(for L2, L3 applications)

Metric: Bus Factor & Reusability

… *"number of team members that have to disappear from a project before the project stalls due to lack of knowledgeable or competent personnel"*, Wikipedia

GitHub Tooling Example:





https://www.playitstartup.com/

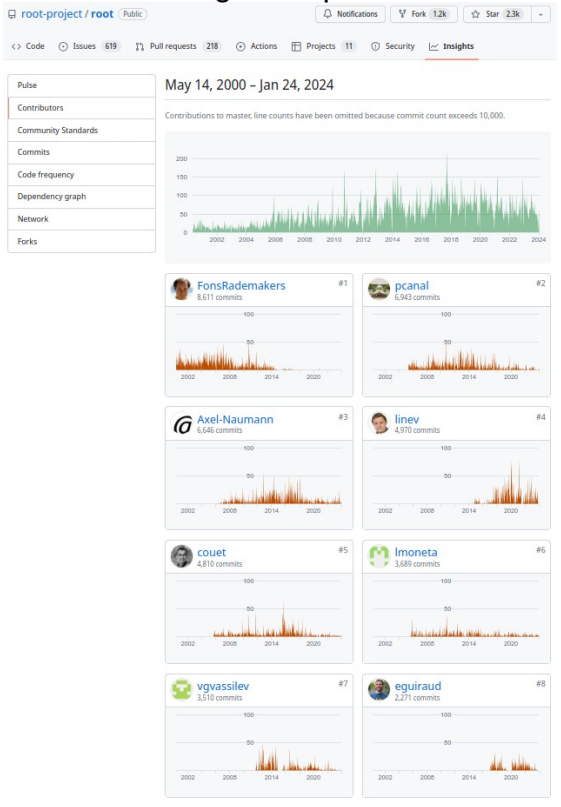
Goal: bus factor ≥ 3

(for L2, L3 applications)

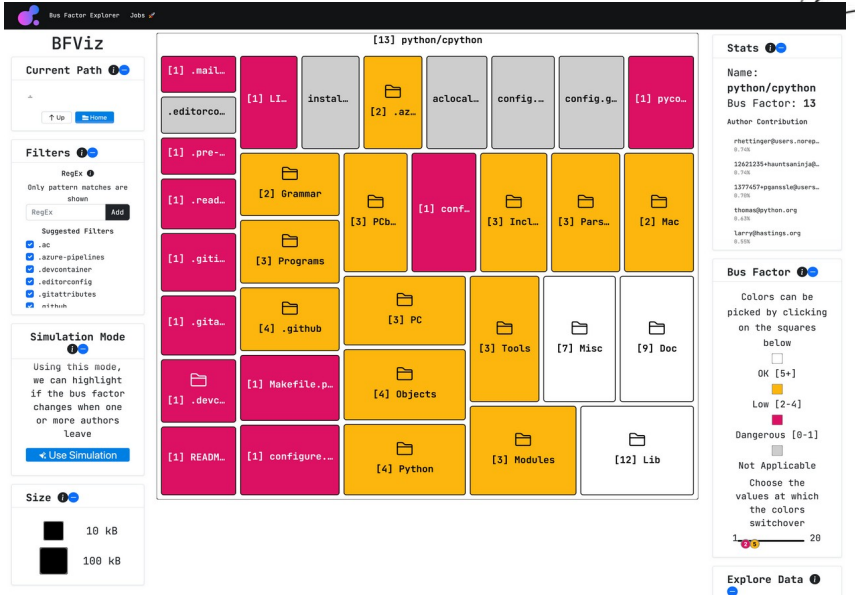
Metric: Bus Factor & Reusability

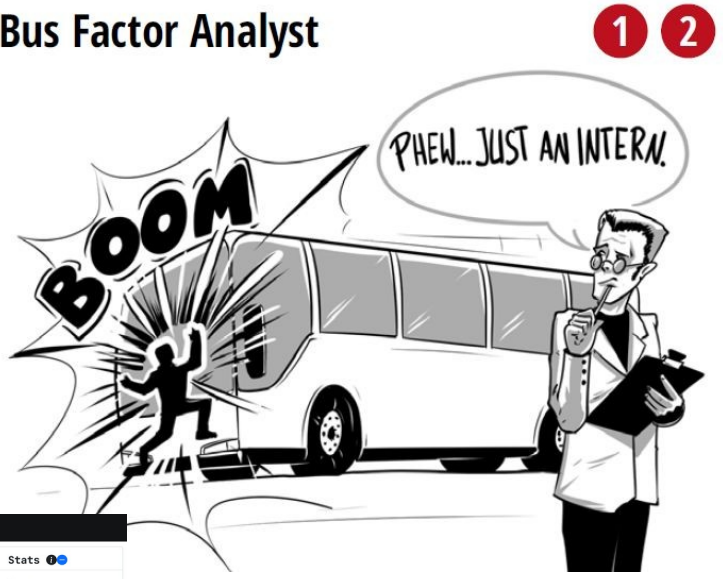
… *"number of team members that have to disappear from a project before the project stalls due to lack of knowledgeable or competent personnel"*, Wikipedia

GitHub Tooling Example:



GitHub Tooling Example (by JetBrains):





https://www.playitstartup.com/

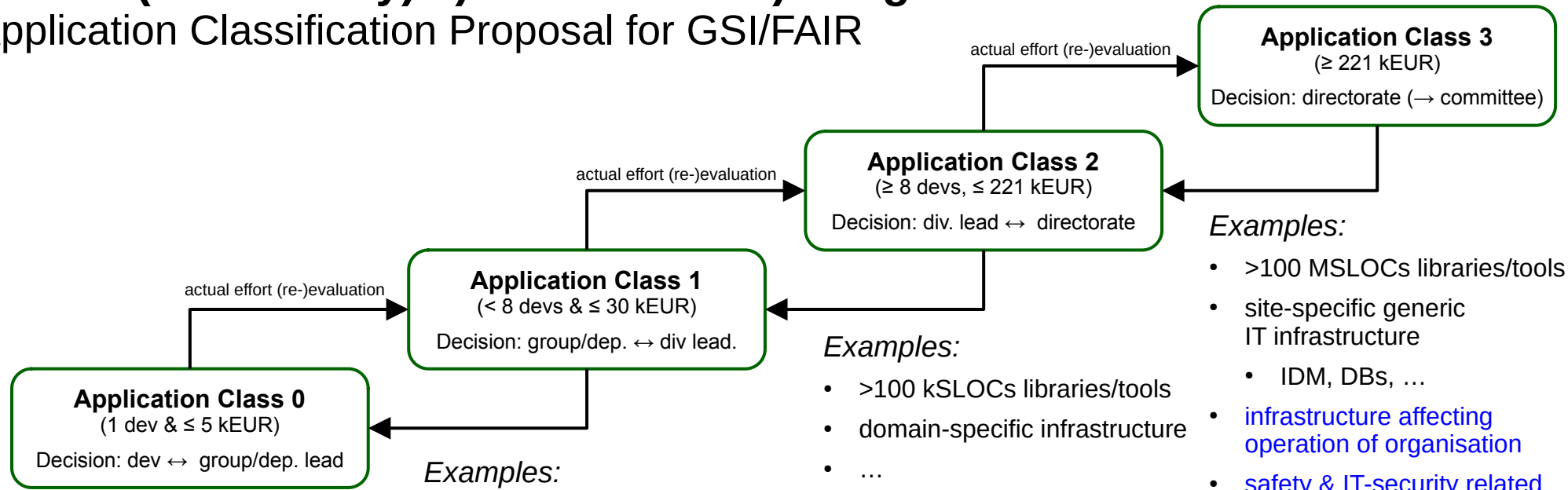
Goal: bus factor ≥ 3

(for L2, L3 applications)

FAIR GSI

HowTo (realistically) a) assess and b) mitigate Risks related to RSE

Application Classification Proposal for GSI/FAIR



Examples:

- < 2 kSLOCs analysis scripts
- spread-sheet macros
- …
- reproducible within days

Examples:

- < 100 kSLOCs libraries/tools
- …
- reproducible within few engineering months

Examples:

- >100 kSLOCs libraries/tools
- domain-specific infrastructure
- …
- reproducible within few FTEs

Examples:

- >100 MSLOCs libraries/tools
- site-specific generic IT infrastructure
  - IDM, DBs, …
- infrastructure affecting operation of organisation
- safety & IT-security related
- non-open-source
- …

greater scrutiny with greater risks

employee

Central Questions regarding Software Guidelines

- Software must be adaptable to frequent changes
  - must provide value and be fit-for-purpose
  - remain easily adaptable to changing requirements
  - is as much about technology as it is about people
- Total Cost of Ownership
  - HowTo (realistically) ‘assess’ and ‘mitigate’ risks related to RSE at GSI/FAIR
  - Proposal: ‘Application Classification Level’ defining level of scrutiny
    - important: make it public, transparent, and fully open-source as default
- What constitutes 'lean' and 'clean' code? Which aspects should be:
  - enticed – Ethics, Netiquette, engaging with collaborators, users, and society, ...
  - recommended – GSI/FAIR team standard, simplifying onboarding, cross-departmental support, or
  - must be enforced ↔ mitigating costs & risk for the organisation/divisions as a whole
- For review as an exemplary draft:
  - CORE_DEVELOPMENT_GUIDELINE.md and perhaps also CORE_NAMING_GUIDELINE.md
  - What about code-formatting? Some “experiments” at https://github.com/fair-acc/graph-prototype
- Missing aspects? Details? Your constructive feedback is required …

That’s all ...


The King is conducting a survey - how do you feel about C++ UG meetings?
Oh, I can't complain.