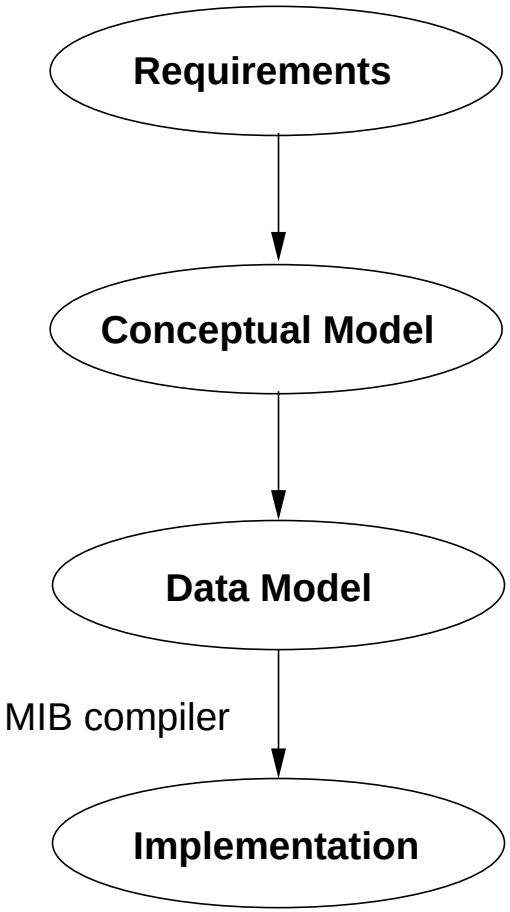
Reverse Engineering Internet MIBs

Jürgen Schönwälder

Computer Science Department
Technical University Braunschweig
Bültenweg 74/75
38106 Braunschweig
Germany

Tel.: +49 531 391 3289
Email: <schoenw@ibr.cs.tu-bs.de>
Web: <http://www.ibr.cs.tu-bs.de/~schoenw/>

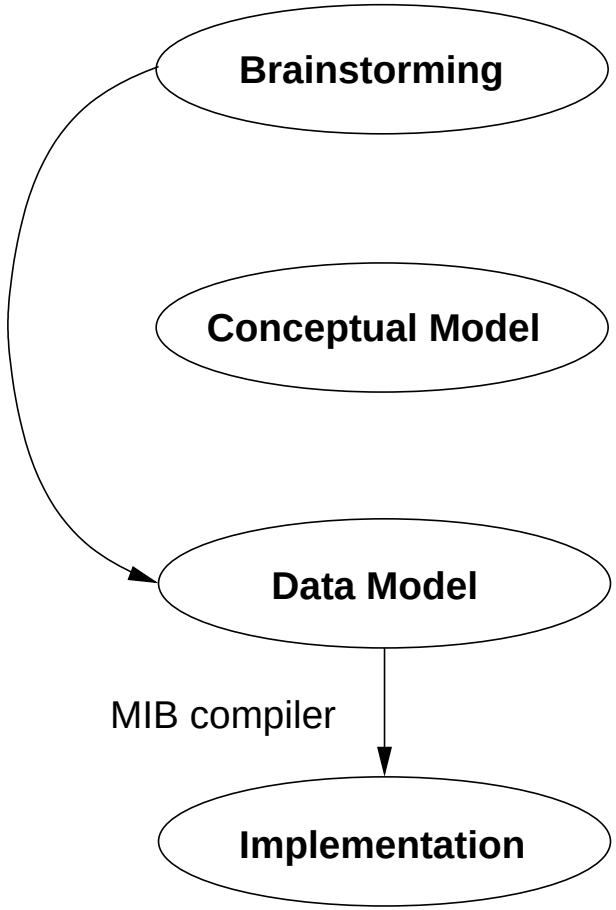
MIB Engineering Process



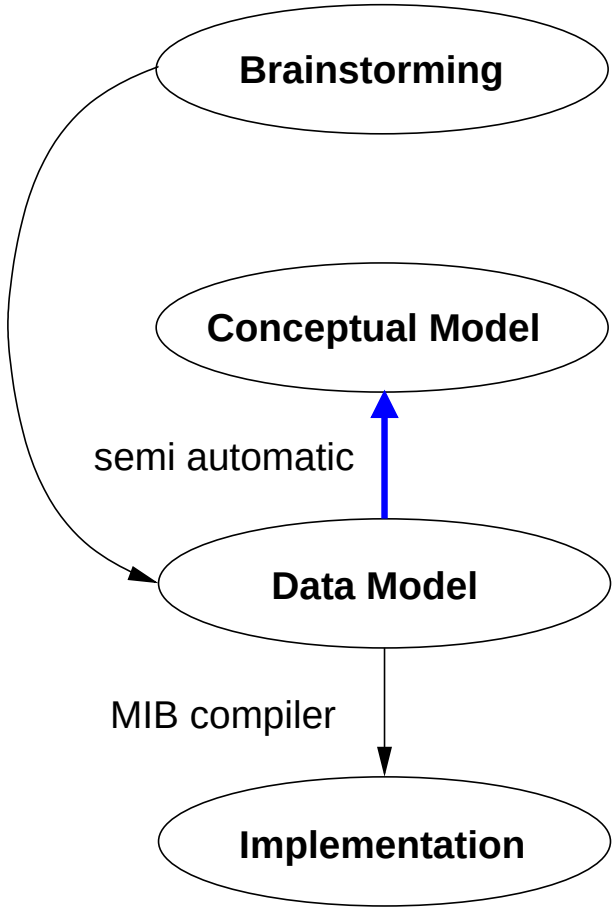
Explicit conceptual models...

- improve consistency (important when authors change)
- simplify quality assurance processes (MIB police)
- enable more efficient and extensible applications
- reduce long-term software maintenance costs
- are generally useful for educational purposes
- simplify integration into more comprehensive models

Reality is not always nice...



Reality is not always nice...



- Is it possible to reverse engineer conceptual models?
- Can this reverse engineering process be automated?
- What is the best way to represent conceptual models?

Representation of Conceptual MIB Models with UML

1. UML classes representing MIB definitions use the ≪`smi mib class`≫ stereotype.

2. MIB table rows are represented as UML classes.

3. Scalars that are logically bound to MIB tables are shown as class attributes.

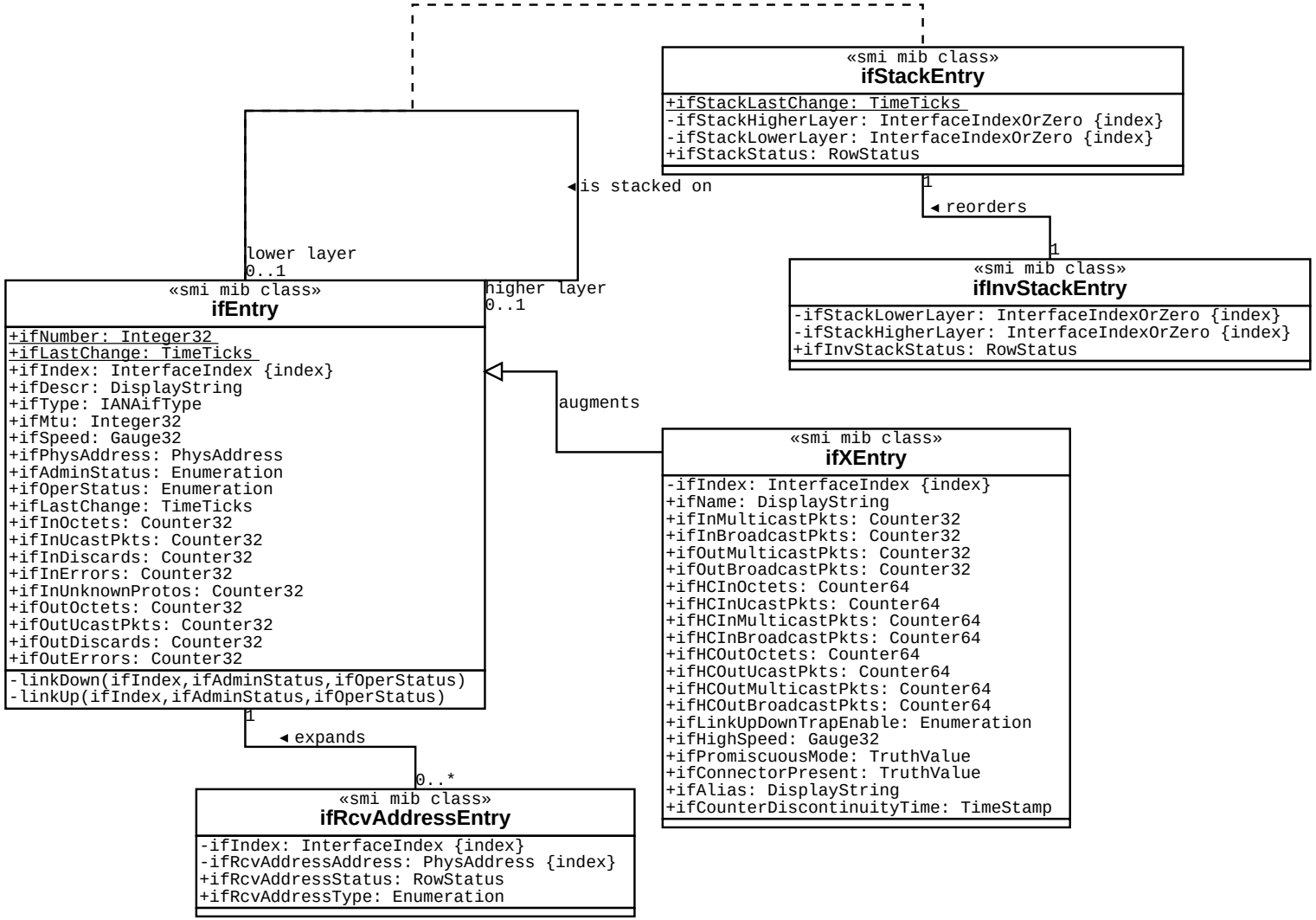
4. Unbound scalars are shown as class attributes of additional auxiliary classes.

5. Notifications are assigned to classes and shown as private operations.

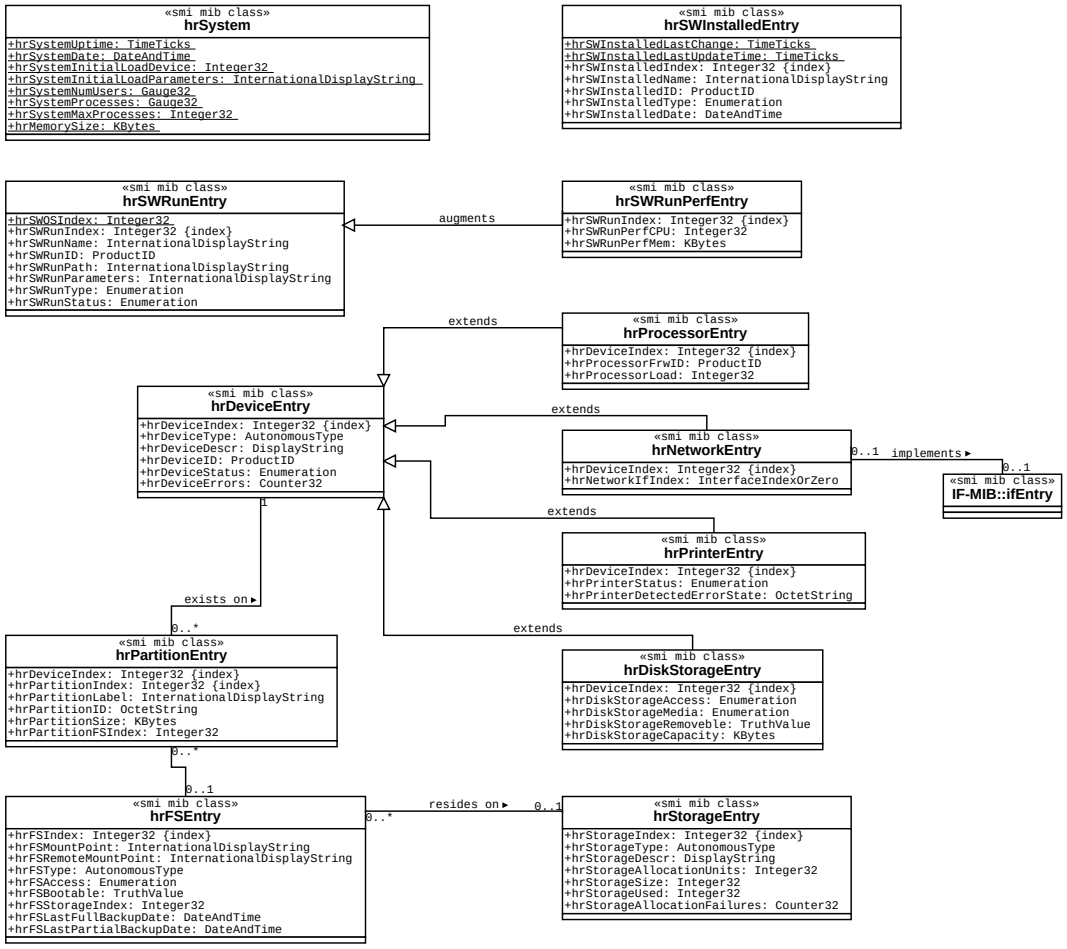
6. Class attributes that identify a class instance are marked with the {`index`} UML property.

7. SMI access modes are mapped to standard UML visibility attributes.

Conceptual Model of the IF-MIB (RFC 2863, RFC 2864)



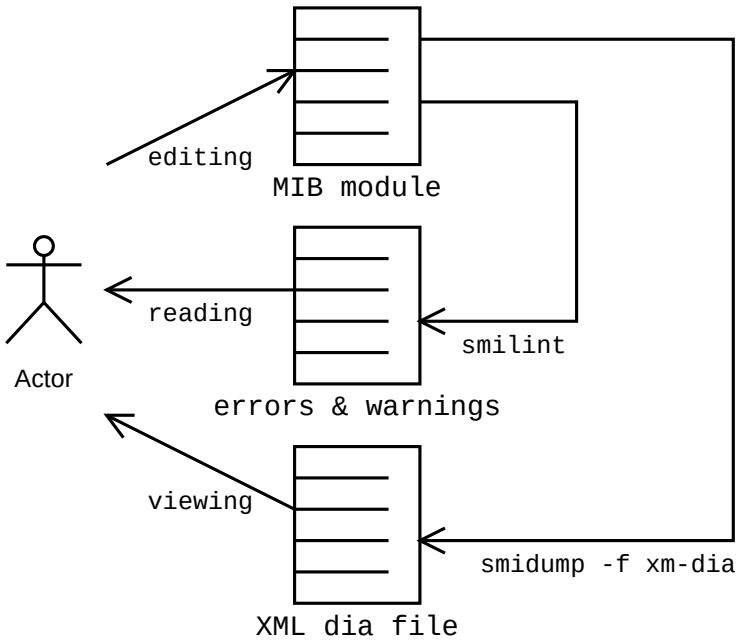
Conceptual Model of the HOST-RESOURCES-MIB (RFC 2790)



Outline of the Reverse Engineering Algorithm

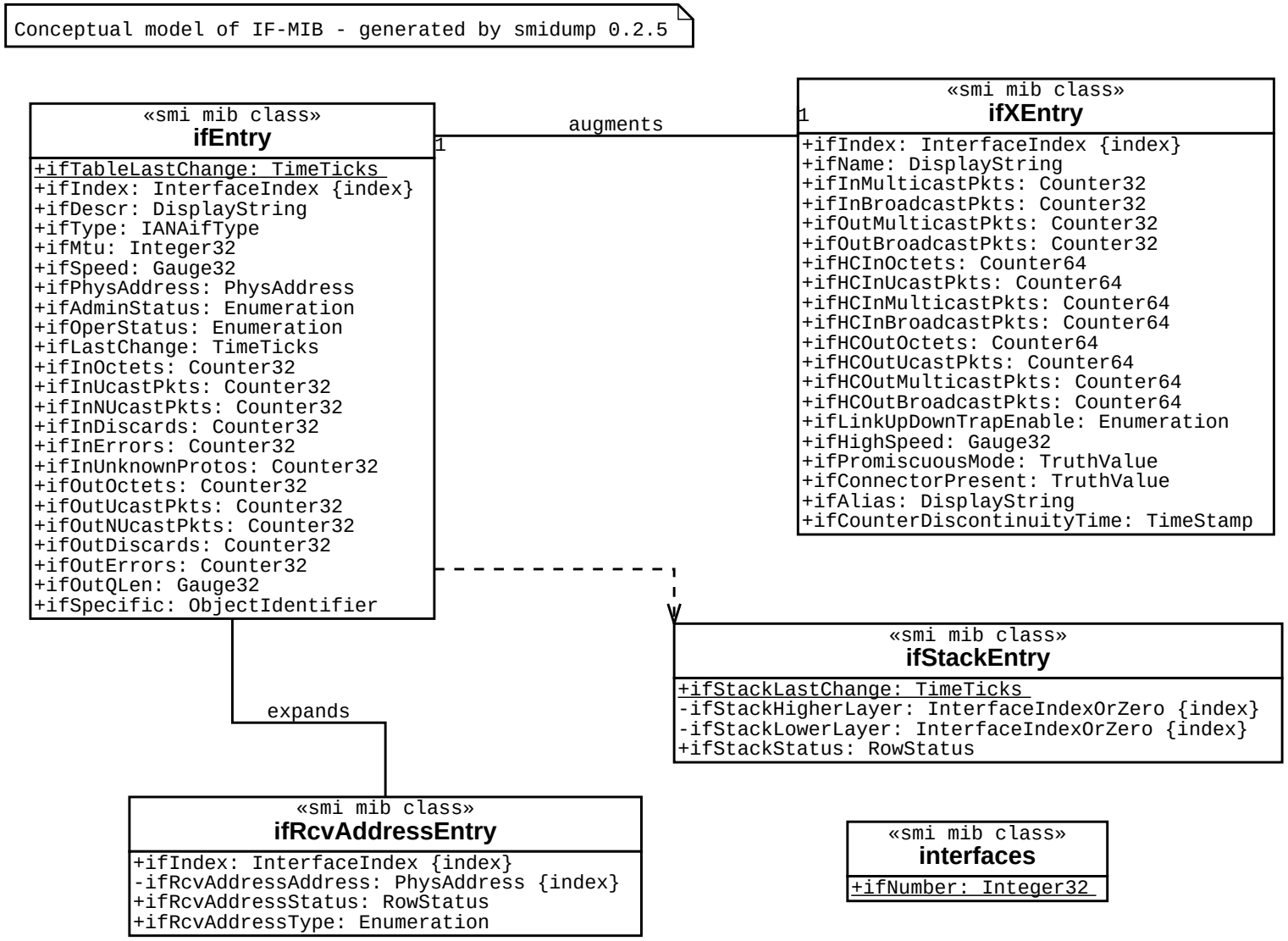
1. Create nodes for all tables and scalars.
2. Create edges for all existence relationships.
3. Reorder edges based on the commonality of normalized names.
4. Create edges for reference relationships by analyzing the usage of index types.
5. Create edges between tables based on common name prefixes.
6. Assign scalars to tables based on the commonality of normalized names.
7. Group nodes representing scalars with a common parent.
8. Create edges representing dependency relationships for tables which only contain “supporting objects” (`RowStatus`, `StorageType`).
9. Create edges for reference relationships by analyzing object names (*Index, *Pointer).
10. Assign notifications to nodes based on the mandatory objects.

Implementation

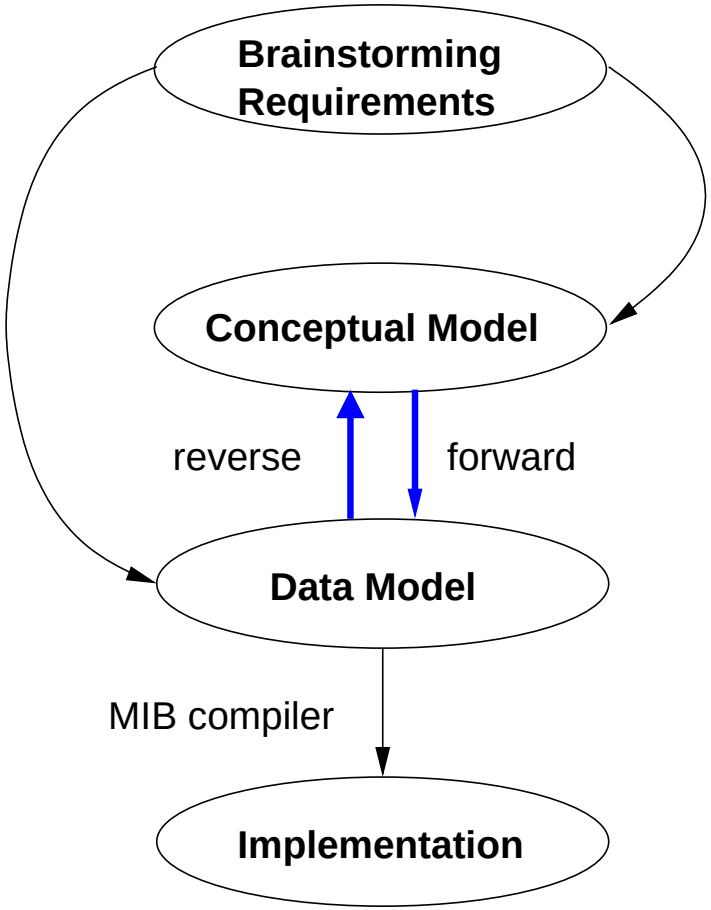


- Implemented in C on top of the `libsmi` and integrated into the `smidump` MIB compiler.
- Produces input for the `dia` UML editor.
- Simple layout algorithm which usually requires manual cleanup.
- Note that layout contains semantics and is hard to automate.
- Probably need ways to tweak the heuristics.

Output for the IF-MIB



Potential Future Work



There are several ways to make things even more useful...

- support round-trip engineering
- allow changes in the conceptual and the data model
- need good algorithms to identify and integrate changes that do not disturb engineers
- perhaps define metrics to estimate complexity

Thanks for listening!

Any questions?