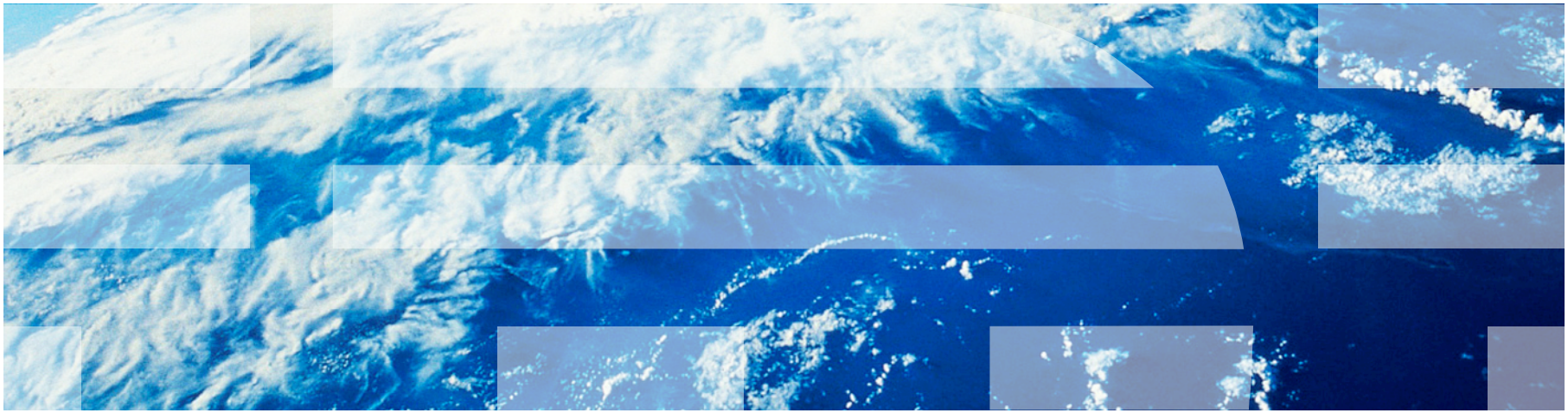
Building a Problem Determination Database

Carolyn Norton, IBM



The Question

What are the characteristics of a good knowledge base, one that lends itself to easy population with, analysis of, and generation of, useful symptom definitions?

What makes a good programming language?

- Speed – of execution, of compilation, of writing
- Expressiveness – does it suit the problem domain?
- Readability
- Language, Framework, Runtime
- Reuse --
  - What don't you have to implement yourself ? libraries, services
  - Can you make existing pieces "fit together" easily
  - Writability and Readability
- Simplicity
- Definiteness
- Orthogonality
- Expressiveness
- Implementability
- Efficiency
- Principle of Least Astonishment

What makes a good symptom definition tool?

- Easy Observations
  - The collection produced needs to be “useful enough, often enough”
    - critical mass
  - The “natural domain” should be
    - Problems that are not too hard.
    - Problems that are not too easy.
    - Problems that really happen. And happen often.
  - It should be easy for authors to
    - Formulate a rule
    - Start to define rules
    - Define a rule
    - Test a rule
    - Deploy a rule
  - Make it easy for users to
    - Find your tool
    - Decide when to use your tool

What makes a good symptom definition tool?

- More Observations
  - The “rules” should be persistent.
    - Over releases, not just over time.

- Open Questions
  - Start with problems or technology?
  - How do you define “rules”? Programmatically? Manually? Systematically? Open-source?

Backup

- Scoping the project
  - Aim to be “Useful enough, often enough”
    - This is NOT a long-tail space. “Often” is good enough. 80-20 Rule.
    - What do computers do well/quickly and people do poorly/slowly?
    - Can your tool be useful even when it’s not?
  - Aim to “Prevent the most customer pain with the least effort.”
    - Not too hard, not too easy.
    - T-shaped knowledge?
    - Beware of single-release symptoms. Added-value, not core value.
  - Consider the natural domain…
    - You get what you ask for.
    - Start with problems, not technology.

- How will you create symptom definitions?
  - Programmatically?
  - Manually?
    - Systematically? By whom? How do you identify the right authors?
    - Open-source?
  - Semi-automatically?

- Make it easy for authors to
  - Formulate a symptom definition
    - When does it NOT apply?
    - Which versions does it apply to?
    - Beware of rules that are easier to describe than code.
  - Start to define rules
    - What tooling is needed? How hard is it to install? How hard is it to learn?
    - Consumability is key!
  - Define a rule
    - Can you partially automate the process?
  - Test a rule
  - Deploy a rule

- Make it easy for users to
  - Find your tool
  - Decide when to bother with your tool