# Assessing the Impact of Crosscutting Concerns

Marc Eaddy

Alfred Aho

Columbia University

# Outline

- Importance of concern analysis
- Crosscutting = Scattering
- Relationship between concerns and other artifacts
- Metrics for quantifying this relationship
- Thoughts on the concern assignment problem

# Concerns

*Every line of code exists for a reason*

- A *concern* is anything that impacts the implementation of a program
  - Requirements, features, design models, policies, coding guidelines
- Important to understand concerns…
  - Why the program is the way it is
  - How the program will evolve and why
- …and how they are implemented
  - How easy it is to create and evolve the program

# Concern analysis

## What are the concerns?

- Concerns as first-class program abstractions

## Where are they in the code?

- Concern mapping
- Manual techniques – Concern Graphs, Software Plans, removal-dependency analysis, …
- Automated techniques – Information retrieval, execution tracing, program analysis, ...

## How were the concerns implemented?

- Concern metrics
- Impact on code quality

## How to best implement concern?

- Impact on modularity, evolvability, maintainability, etc.
- OOP, AOP, FOP, open classes, side classes, etc

# My research

## What are the concerns?

- Concerns as first-class program abstractions

## Where are they in the code?

- Concern mapping
- Manual techniques–Concern Graphs, Software Plans, **removal-dependency analysis**,…
- Automated techniques – Information retrieval, execution tracing, program analysis, ...

## How were the concerns implemented?

- **Concern metrics**
- **Impact on code quality**

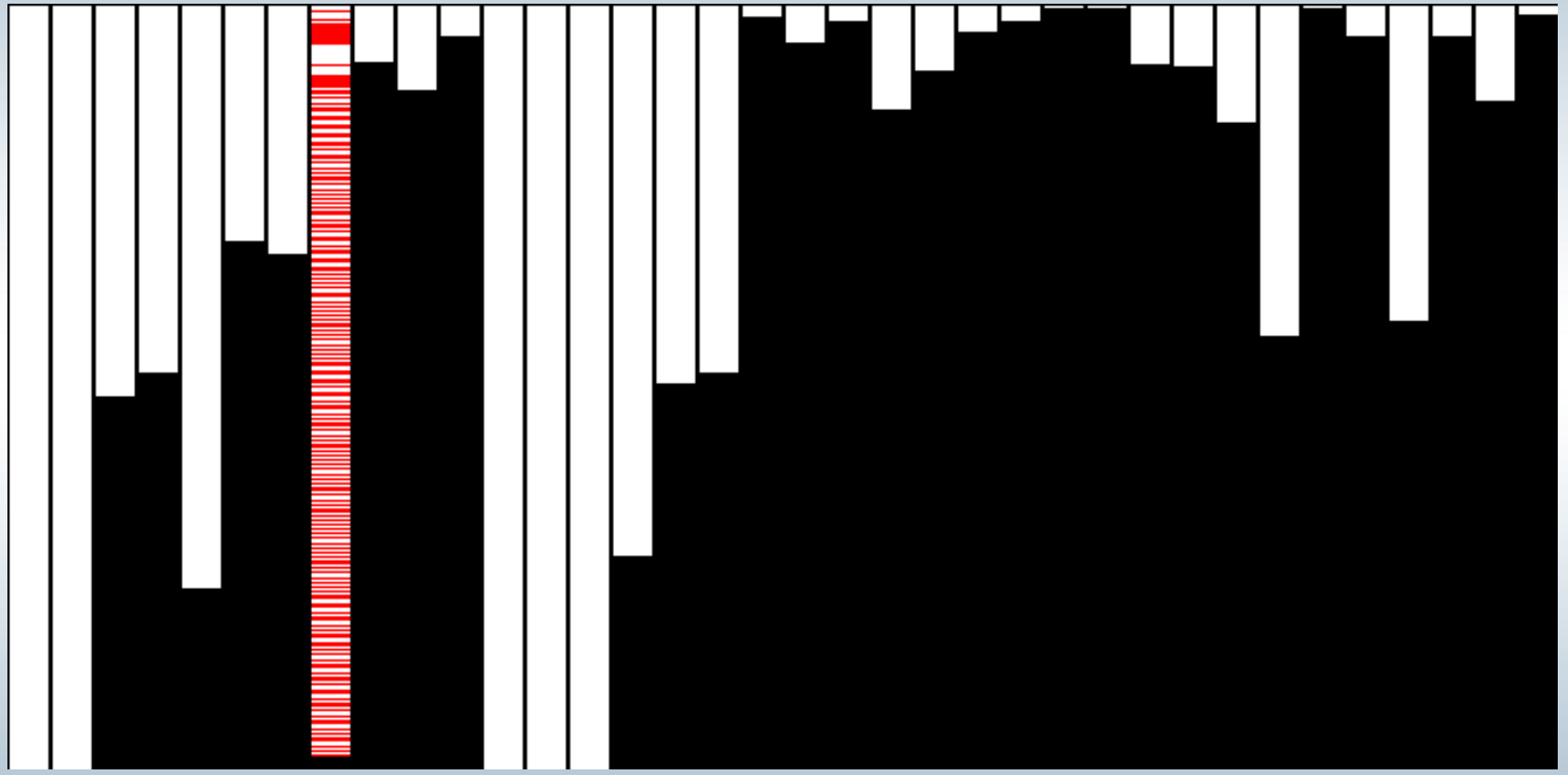
## How to best implement concern?

- Impact on modularity, evolvability, maintainability, etc.
- OOP, AOP, FOP, open classes, **side classes**, etc

# Crosscutting concerns

- A concern is *crosscutting* if its implementation is scattered across multiple program elements
- Crosscutting concerns are often tangled with code related to other concerns
  - Is tangling required for crosscutting?
  - Tangling = coupling?

# Goblin: Game AI



- Game AI is “localized”

# Goblin: Collision detection

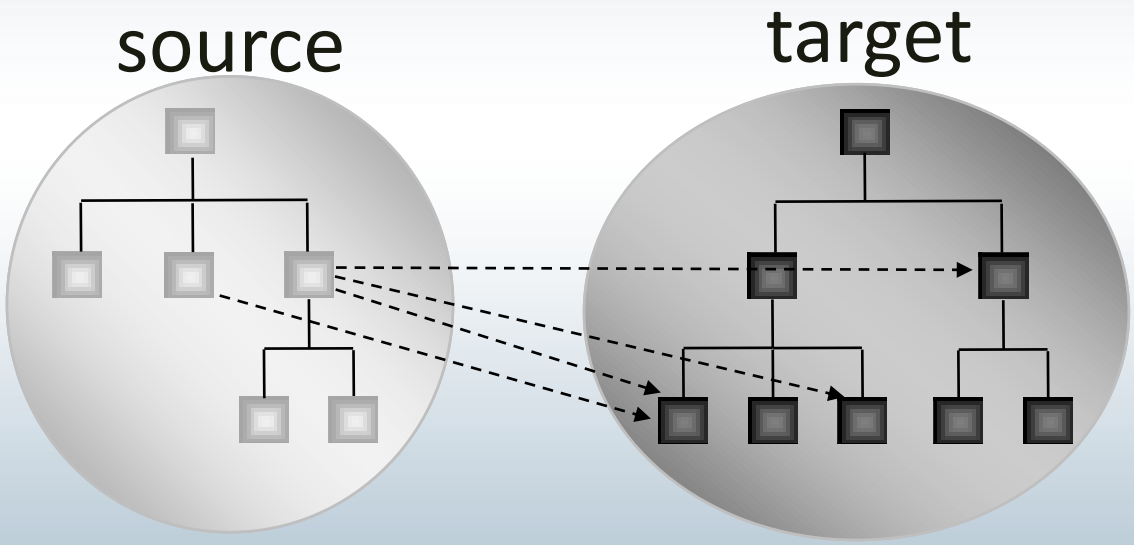
- Collision detection is “scattered” (“crosscutting”)

# Crosscutting hurts modularity

- Crosscutting concern
  - More scattered → Less modular
  - Harder to implement, evolve, maintain, and understand
- Affected program elements
  - Concerns not separated—coupled instead

# Concern model

- Source = concerns or artifacts
- Target = artifacts
  - Requirements, design elements, program components, …




- Traceability throughout software development lifecycle
- Multiple granularities and abstraction levels

# Crosscutting impact insufficiently quantified

- Existing concern metrics insufficient
  - Measure *presence* of a concern, not *degree*
  - *Diffusion metrics* (Garcia et al.), *spread metric* (Lai and Murphy, Revelle et al.), etc.
  - Cannot distinguish between a concern split 50-50 and 99-1
  - Cannot distinguish impact of some refactorings
- Concerns are informal (subjective) and incomplete
  - Conclusions are questionable

# Concern metrics

- Adapted and extended *closeness metrics* to measure crosscutting
- *Concentration* – Amount of a concern's implementation that is contained in a specific component

$$CONC(s,t) = \frac{SLOCs\ in\ component\ t\ related\ to\ concern\ s}{SLOCs\ related\ to\ concern\ s}$$

- *Dedication* – Amount of a component's implementation that is related to a specific concern (extended to handle tangled concerns)

$$DEDI(t,s) = \frac{SLOCs\ in\ component\ t\ related\ to\ concern\ s}{SLOCs\ in\ component\ t}$$

- Closeness metrics provide data points, not insight
- Unwieldy for measuring the impact of a refactoring

W. E. Wong, S. G. Swapna, and R. H. Joseph, "Quantifying the closeness between program components and features," *Journal of Systems and Software,* 2000.

# My metrics: DOS

- *Degree of scattering* – Measures the distribution of a concern's implementation across multiple components

$$DOS(s) = 1 - \frac{|T| \sum_t^T \left( CONC(s,t) - {}^1/_{|T|} \right)^2}{|T| - 1}$$

- *Average DOS* – Overall modularity of concerns
  - Summarizes amount of crosscutting present
- More insightful than traditional metrics
  - "class A is highly coupled" vs. "feature A is hard to change"

# My metrics: DOF

- *Degree of focus* – Degree to which a component's implementation relates to multiple concerns

$$DOF(t) = \frac{|S| \sum_{s}^{S} \left( DEDI(t,s) - {}^{1}/_{|S|} \right)^2}{|S| - 1}$$

- *Average DOF* – Overall separation of concerns
- Reflects amount of tangling

# Concern assignment

- Existing manual assignment techniques ambiguous
- What concerns should be associated with `Main`? `System.String`?
  - Avoid implying every concern is crosscutting
  - What is assessment goal?
- Association based on likely change impact
  - Things likely to change should be easy to change (Parnas)
  - Solves problem for `Main`, but still too vague
  - Cannot consider all possible changes
  - A change to a concern can still impact all elements

# Removal-based dependency assignment

- Associated based on impact of *eradicating* a concern
  - Goal of *software pruning*
  - Not just disabling the concern
- Approximates other types of changes
  - Modifying existing concerns
  - Adding new concerns
  - Empirical evidence needed

# Concluding remarks

- Concern analysis is promising and relatively new field
- Need to better understand how concerns are implemented
  - Impact on modularity and code quality
  - Degree of scattering
  - Degree of focus
  - Paper at ACoM‘07
- Need better concern assignment guidelines
  - Ultimately, automated techniques are needed

Thank you!