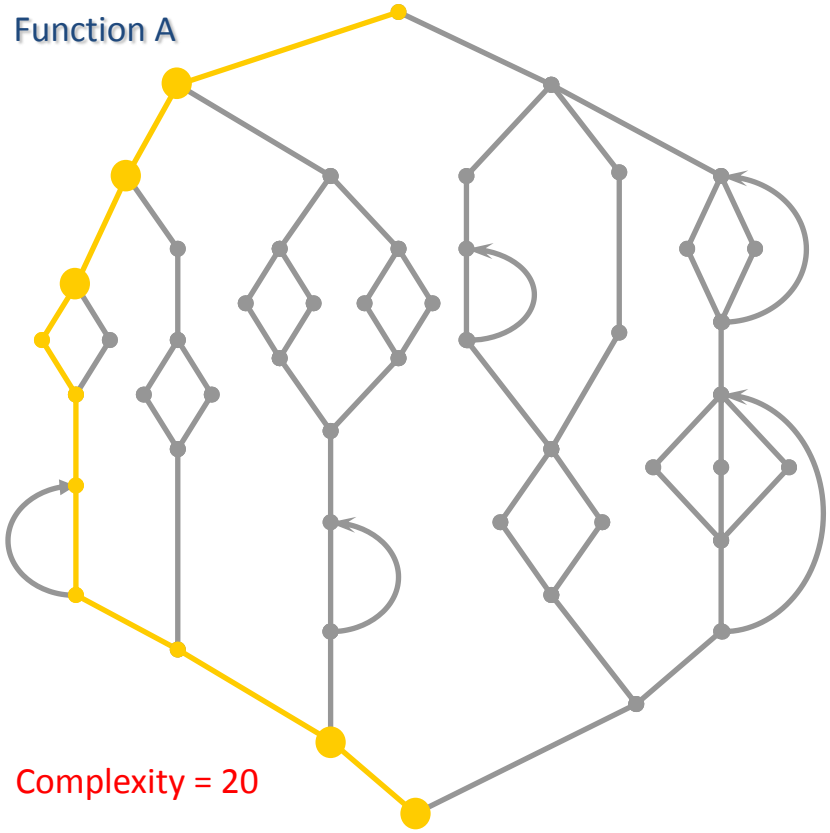
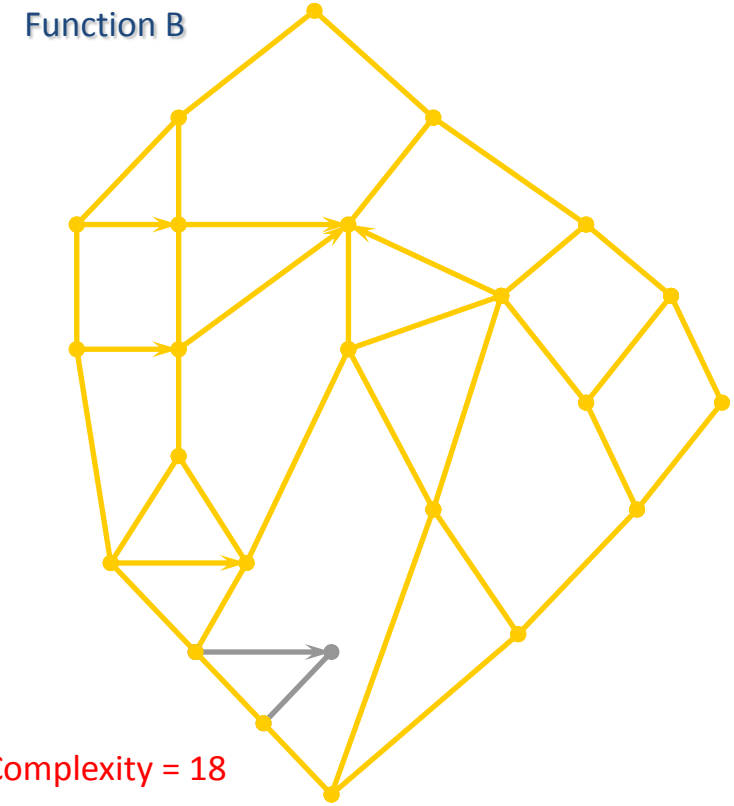
Complexity (decisions) & # lines here are similar, but …



Complexity = 20

Unstructuredness (evg) = 1

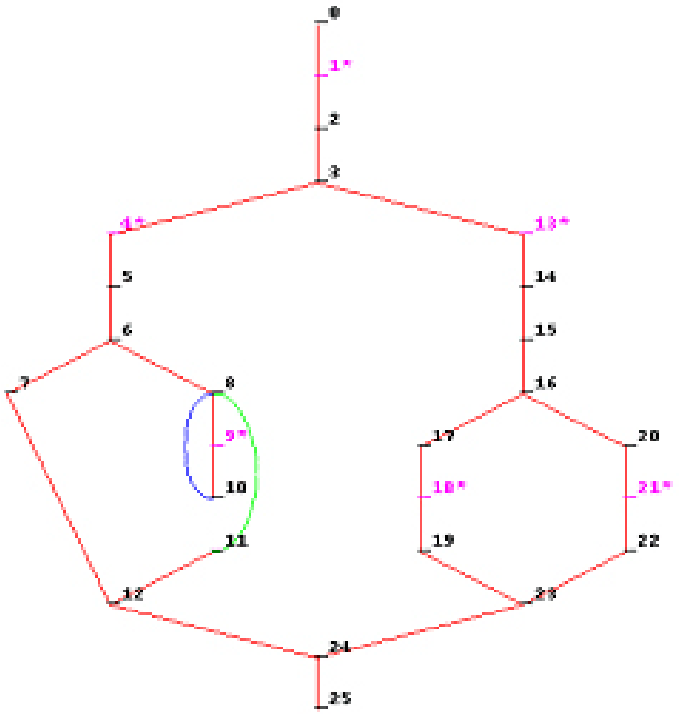


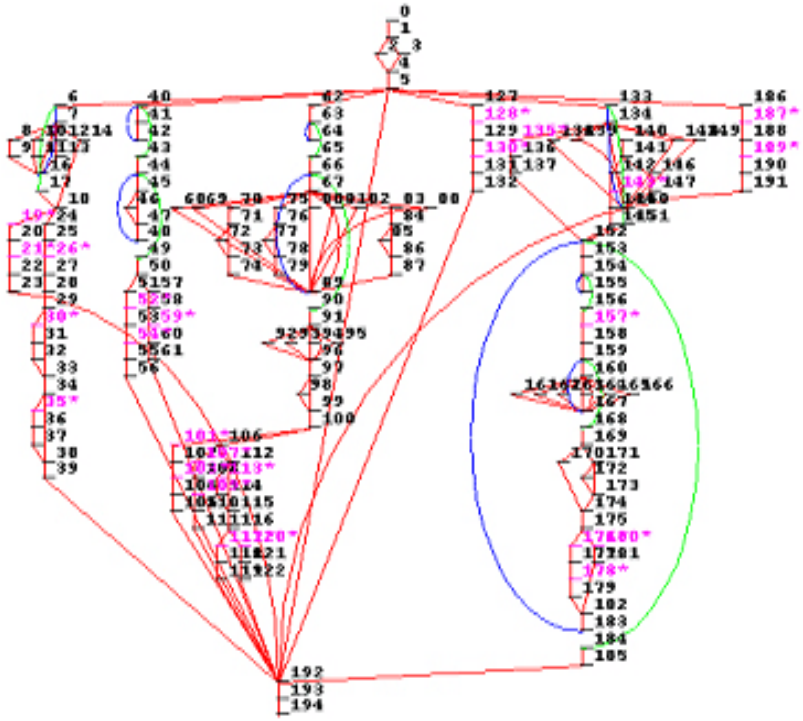
Complexity = 18

Unstructuredness (evg) = 17

So B is MUCH harder to maintain!

For these two modules, which is more likely to have Reliability problem?





This is Cyclomatic Complexity, as an indicator of Reliability.

Pareto Analysis

- At least 80% of run time in less than 20% of code ---- what code?
- At least 80% of errors in less than 20% of code – what code?
- The team will spend 80% of it’s time on less than 20% of code ---- what code?
- Before hand with model, after with profiling

Quality of maintenance

- Programmer productivity varies by 10X
- Maintenance productivity/risk varies by 100X
- The damage a poor programmer can do is only incremental
- The damage a poor maintainer can do is catastrophic
- How to grow and reward good ‘maintainers’?
- Everyone wants to write new code, not find and reuse what’s there

Complexity Metrics Impacting Maintenance

- Cyclomatic Complexity & Reliability Risk
  - 1 – 10 Simple procedure, little risk
  - 11- 20 More Complex, moderate risk
  - 21 – 50 Complex , high risk
  - >50 Untestable, VERY HIGH RISK
- Cyclomatic Complexity & Bad Fix Probability
  - 1 – 10 5%
  - 20 –30 20%
  - > 50 40%
  - Approaching 100 60%
- Essential Complexity (Unstructuredness) & Maintainability (future Reliability) Risk
  - 1 – 4 Structured, little risk
  - > 4 Unstructured, High Risk

Code Attributes by Complexity

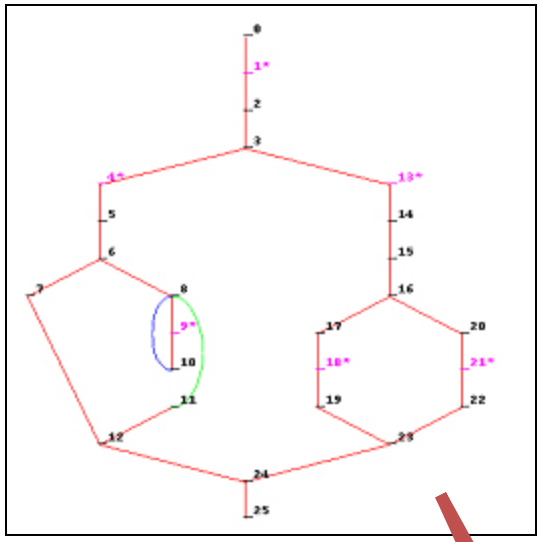
Complexity Metrics Impacting Reliability - Module

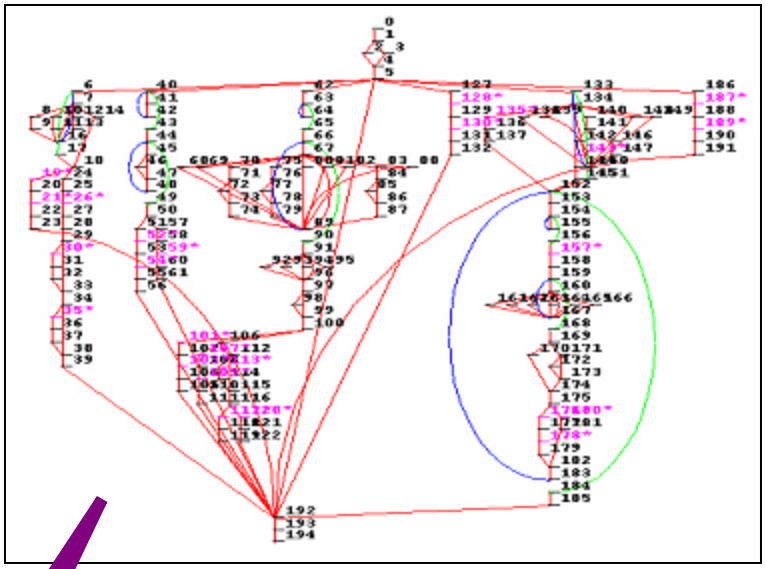
Module Metrics - A Suggested Static Usage

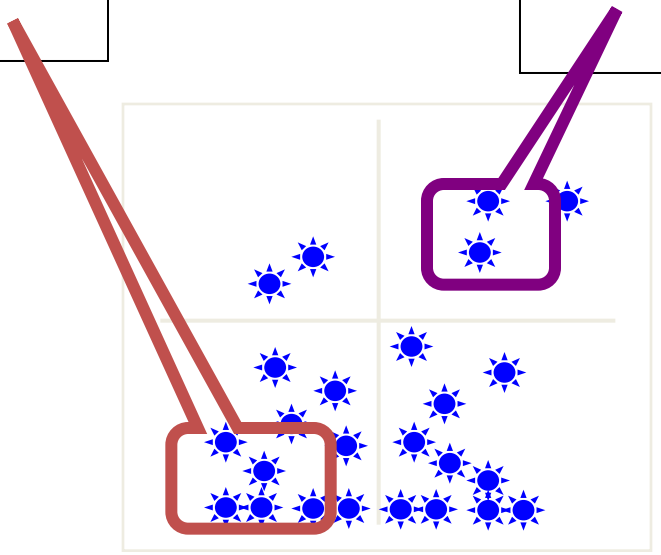
- Outlier Focus - identify most complex modules (# decisions, unstructuredness, etc.), i.e., the outliers, & focus more effort (time, experience of developer, refactoring) on them because they represent the greatest reliability risk

v(g)	ev(g)	iv(g)	gdv(g)	summary
10	1	1	1	small well structured
45	1	1	1	local data switch?
45	1	1	45	global data switch?
45	1	45	1	execution switch?
34	19	28	12	complex & unstructured

Visualizing Complex Code







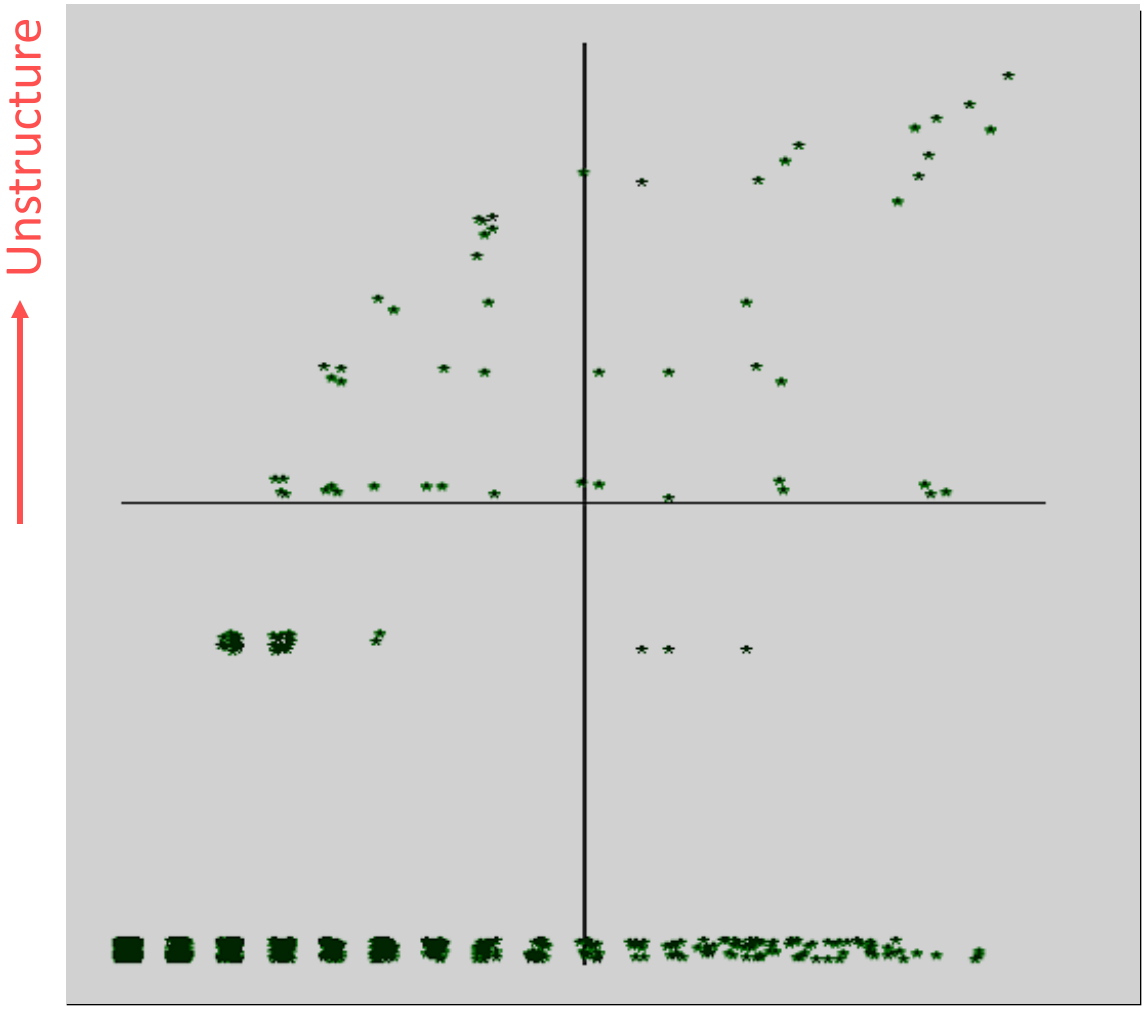
•Simple Code:

Easy to understand, maintain, change restructure, or test

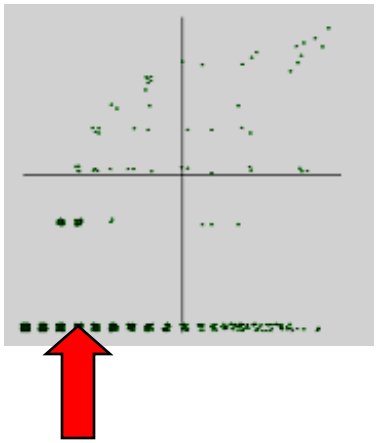
•Complex code:

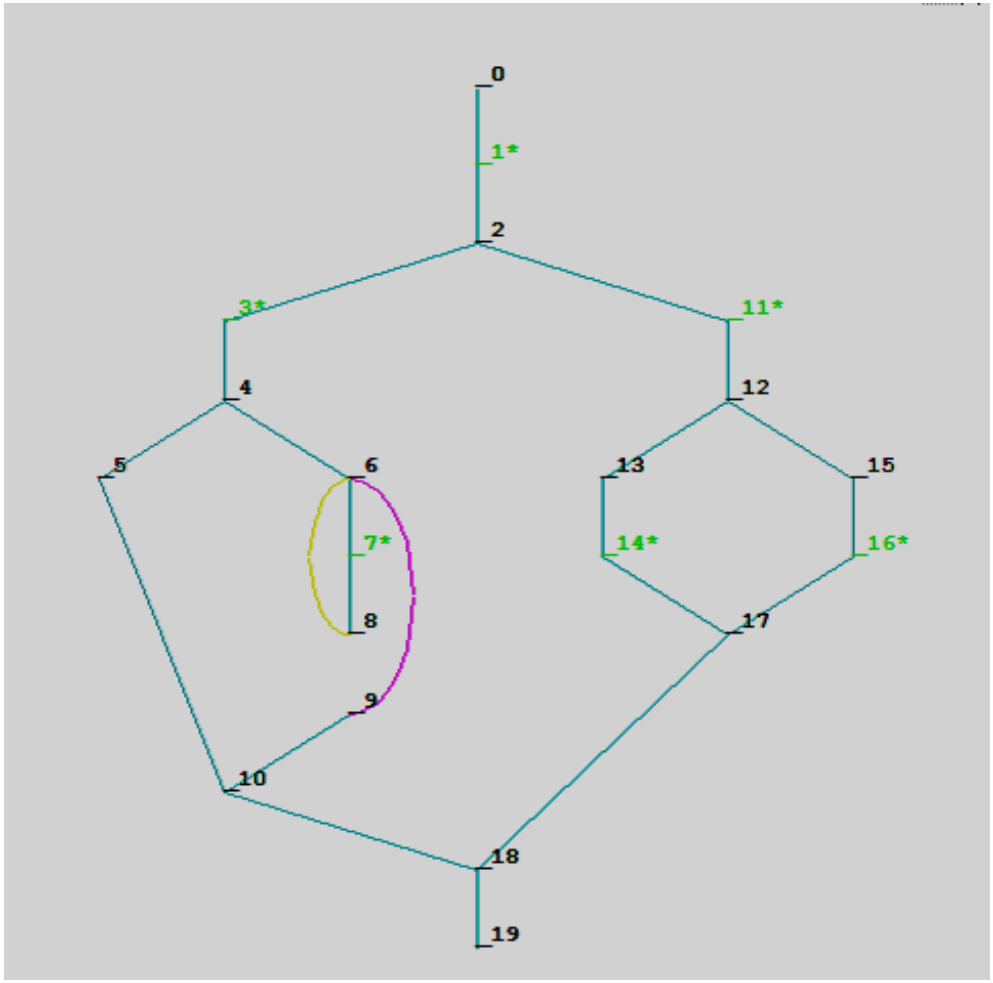
Error prone and hard to understand, maintain, change, restructure, or test

Project Measurement

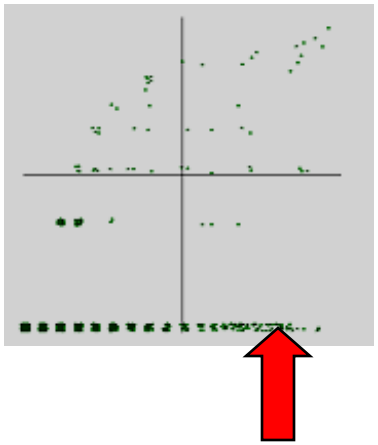


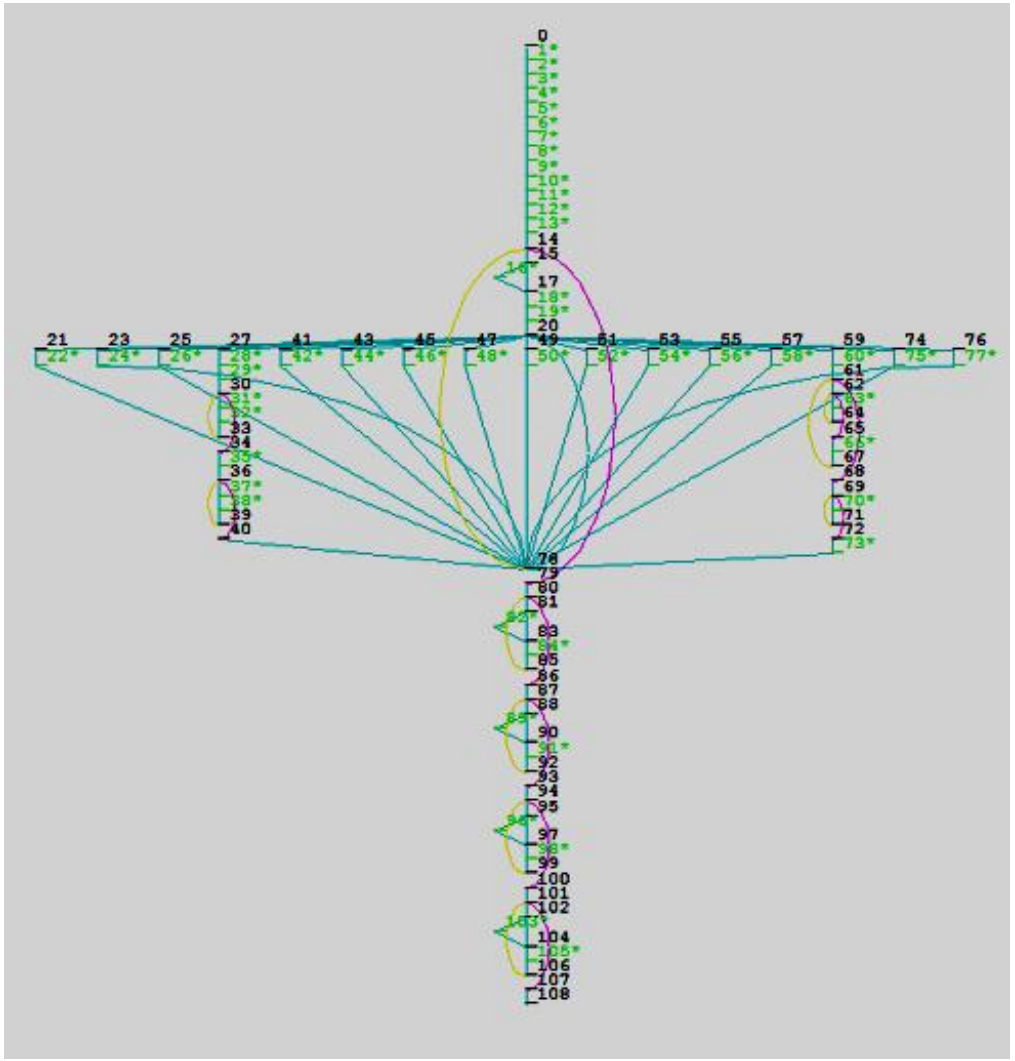
Project Visualisation



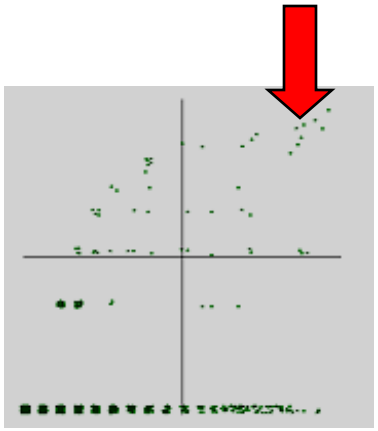


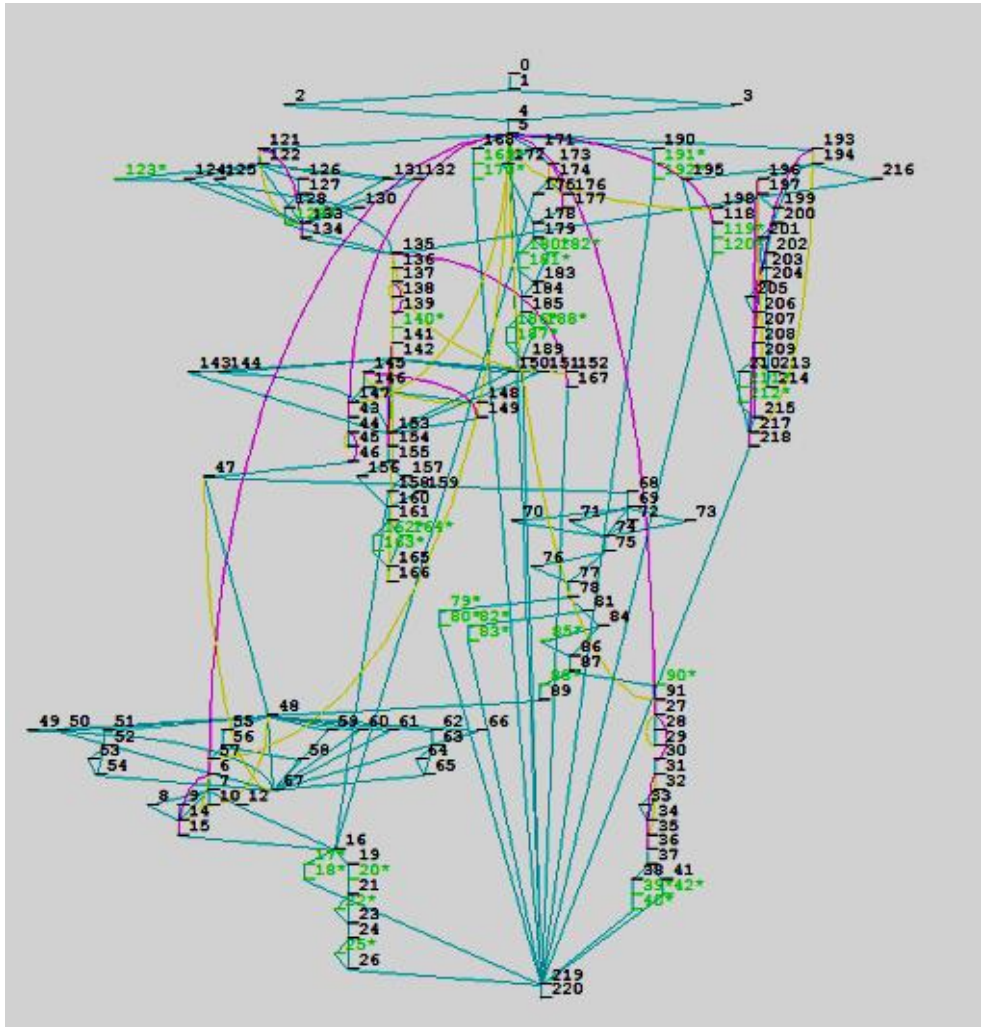
Project Visualisation





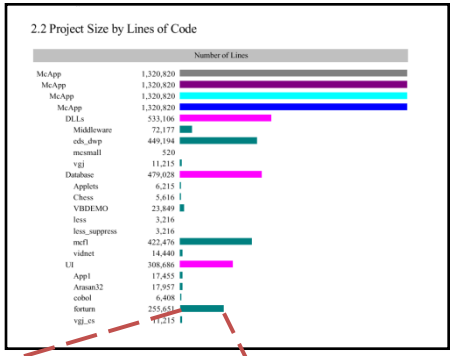
Project Visualisation





Structural Code Analysis - Visualization And Metrics

- Structure CAN be expressed in Understandable terms



At Organization/Enterprise Level

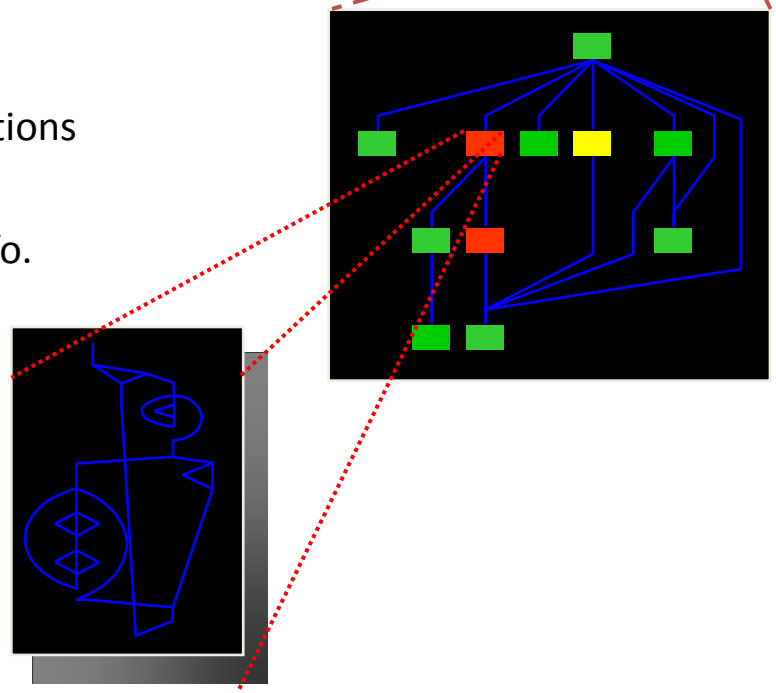
- Multi-level groupings of metrics on structure (size, complexity, maintainability, etc.)

At Component/Design Level

- Structure & class charts, w/ interactions
- Metrics analysis of interactions
- Overlay of static & test coverage info.

At Module (Unit) Level

- Flowgraphs w/ code
- Detailed metrics analysis
- Overlay of static and test coverage info.



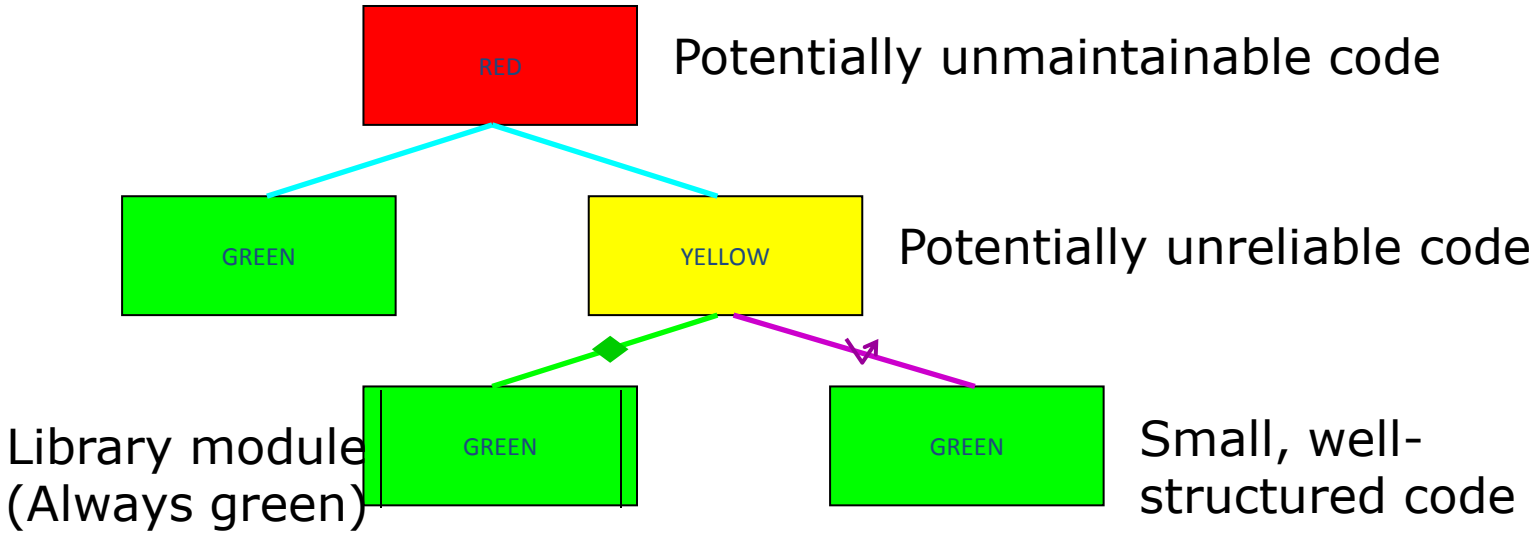
V. McCabe IQ - Applying Tools for Improving Reliability

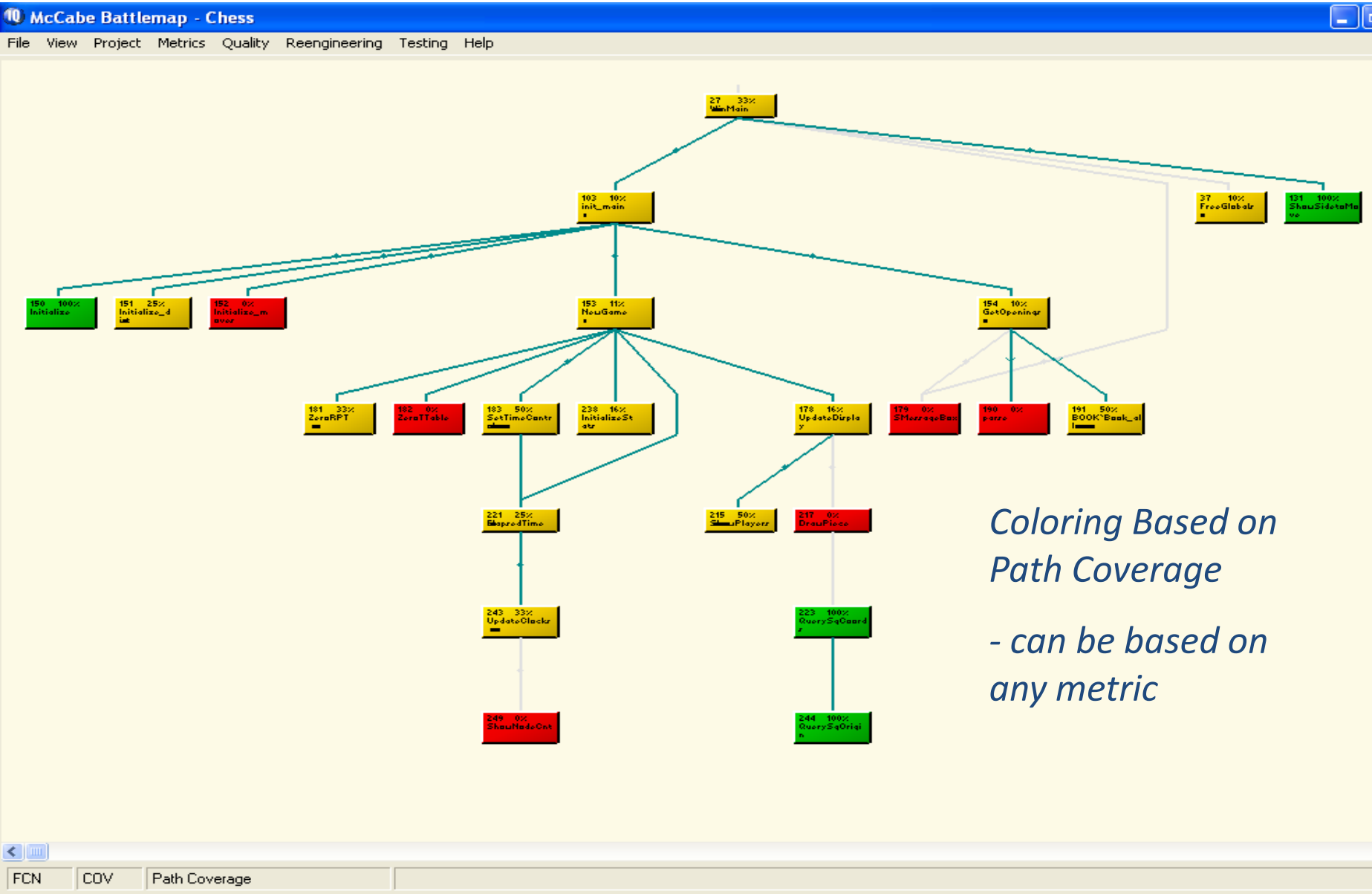
Development Supported By McCabe IQ

Component/Application - Functional Structure Chart

- provides visualization of component design,
  with module calls and superimposed metrics coloring

- is valuable for comprehension

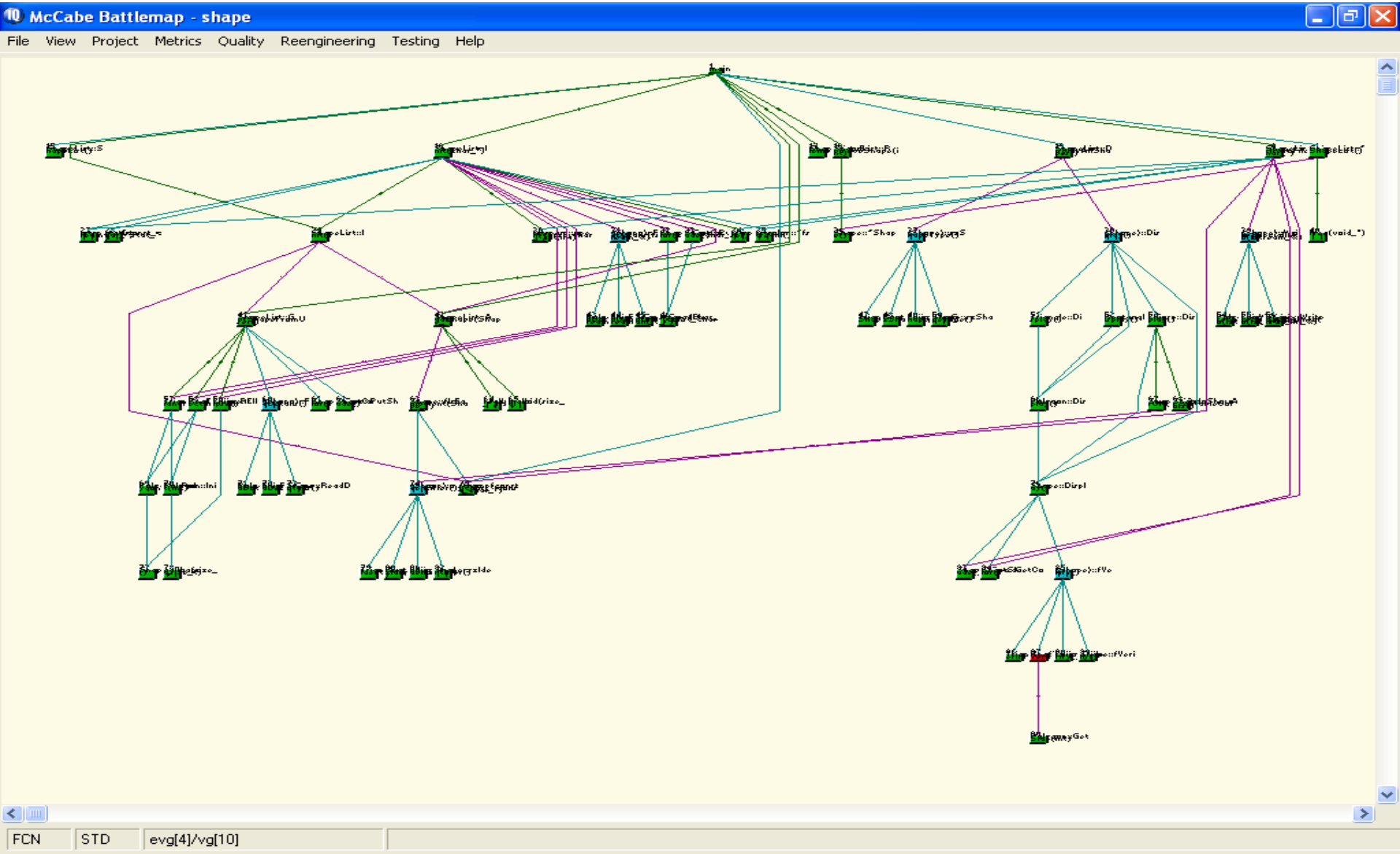




Coloring Based on Path Coverage

- can be based on any metric

V. McCabe IQ - Applying Tools for Improving Reliability



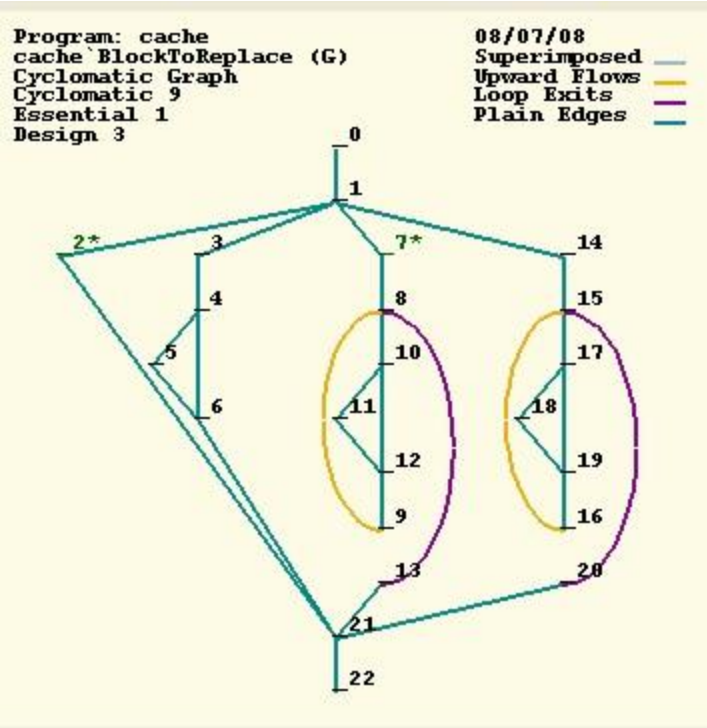
Slicing technology

- Run the transactions you want, the tool highlights executed statements and paths.
- Data slicing, specified data complexity
- Transaction ‘signatures’

Codebreaker

- Two related problems
  - Finding reusable code
  - Finding redundant code

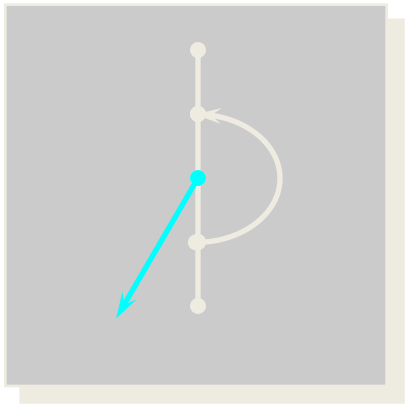
Understand the Control Flow of Code



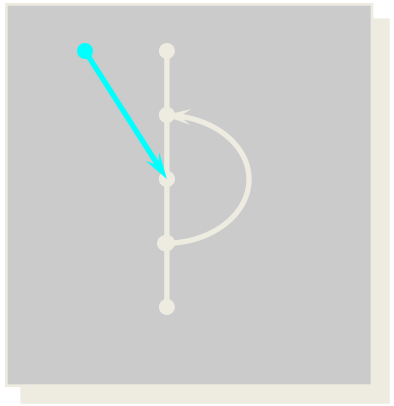
Annotated Source Listing

```
Program : cache
File    : cache.c
Language: c_npp
Module Module
Letter Name                                              v(G) ev(G) iv(G)
------ ----------------------------------------------------------------
   G   cache`BlockToReplace                               9     1     3

170     G0              static unsigned int
171                     BlockToReplace(set_num, cache)
172                     Address set_num;
173                     Cache *cache;
174                     {
175                     /* return appropriate block in set_num to
176
177                        unsigned int replace = 0, lfu, i;
178                        long lru;
179
180     G1                 switch (REPLACEMENT_SCHEME) {
181                              case RANDOM:
182     G2*                           replace = rand() % num_block
183                                   break;
```

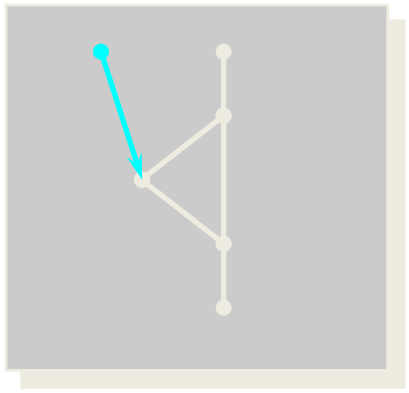
Unstructured Logic



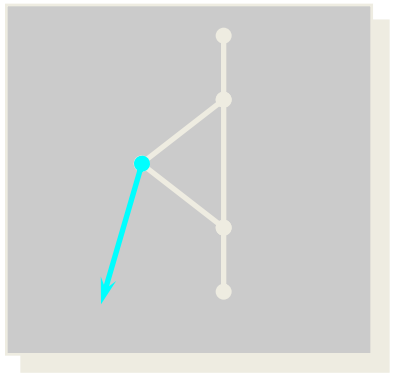
Branching out of a loop



Branching in to a loop

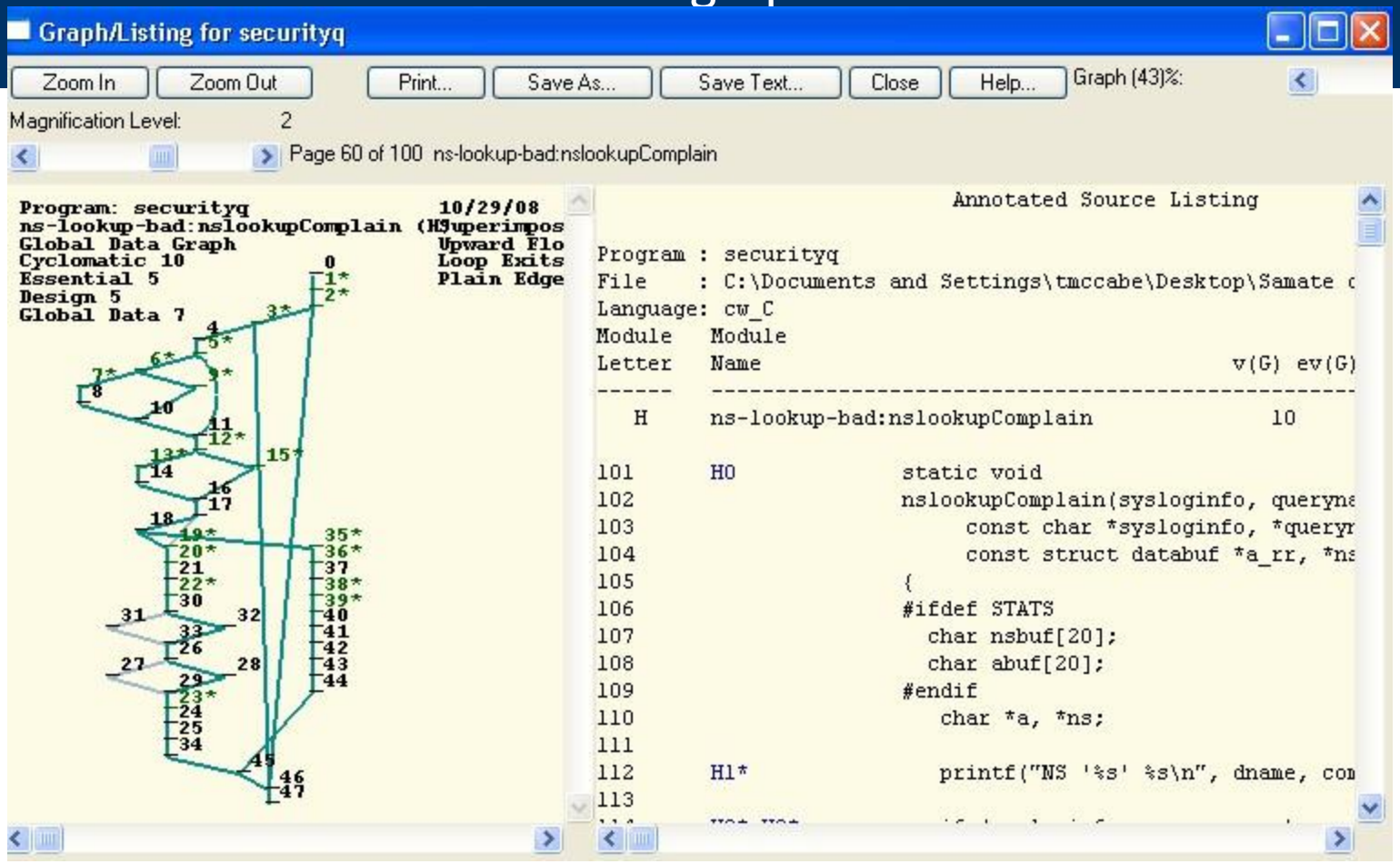


Branching into a decision

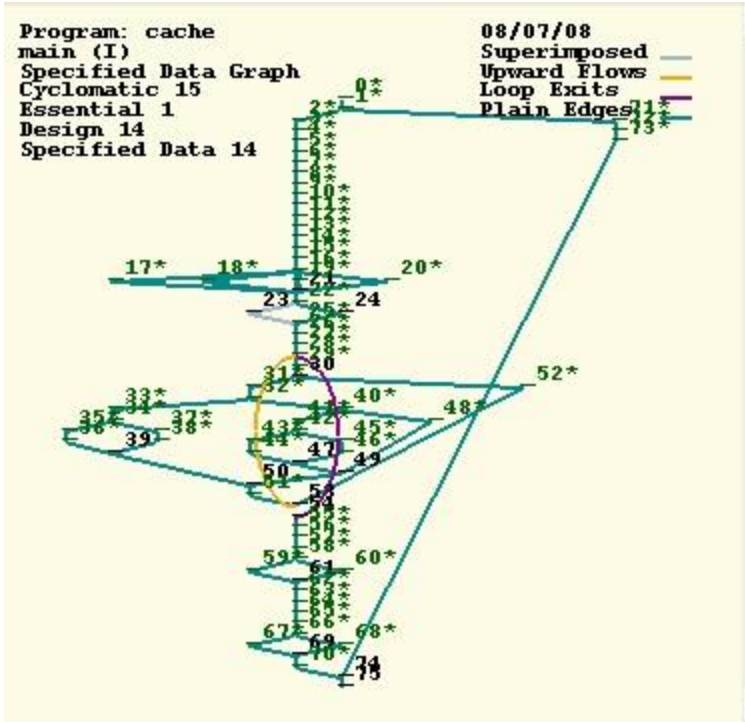


Branching out of a decision

Global Data Flowgraph and Metrics

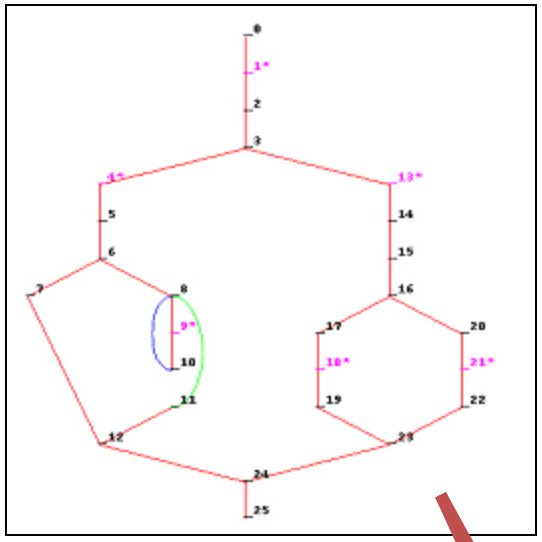


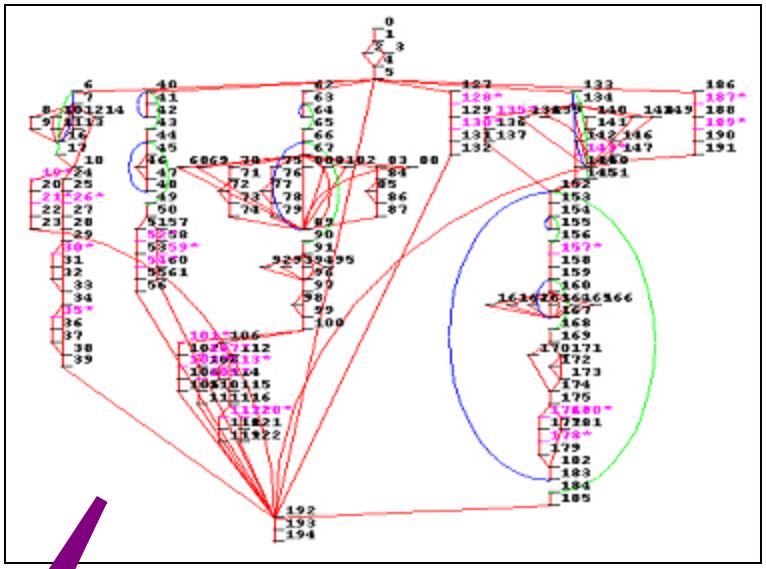
Specified Data Metric and Flowgraph

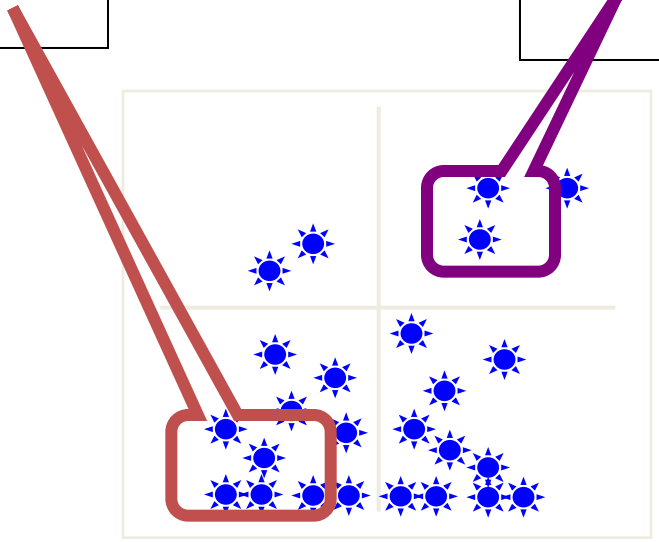



```
                    index_size   = address_size - offset_size;

                    /* BLOCK_FRAME_MASK used to omit block offs
                    BLOCK_FRAME_MASK = 0xffff & (0xffff << offs
I8*                 (void)printf("num_sets_in_cache  = %u\n", n
I9*                 (void)printf("num_blocks_per_set  = %u\n",
I10*                (void)printf("num_words_per_block = %u\n",

I11*                (void)printf("address_size = %u\n", address
I12*                (void)printf("offset_size  = %u\n", offset_
I13*                (void)printf("index_size   = %u\n", index_s
I14*                (void)printf("BLOCK_FRAME_MASK  = %8.8lx\n"

I15*                (void)printf("REPLACEMENT_SCHEME = ");
I16*                switch (REPLACEMENT_SCHEME) {
I17*                      case RANDOM: printf("RANDOM\n"); brea
I18*                      case FIFO: printf("FIFO\n"); break;
I19*                      case LRU: printf("LRU\n"); break;
I20*                      case LFU: printf("LFU\n"); break;
I21                 }

I22* I25*           (void)printf("WRITE_POLICY = %s\n", WRITE_
I23 I24                                                  "Write
```

Visualizing Complex Code







- Simple Code: ***Easy to understand, maintain, change restructure, or test***

- Complex code: ***Error prone and hard to understand, maintain, change, restructure, or test***