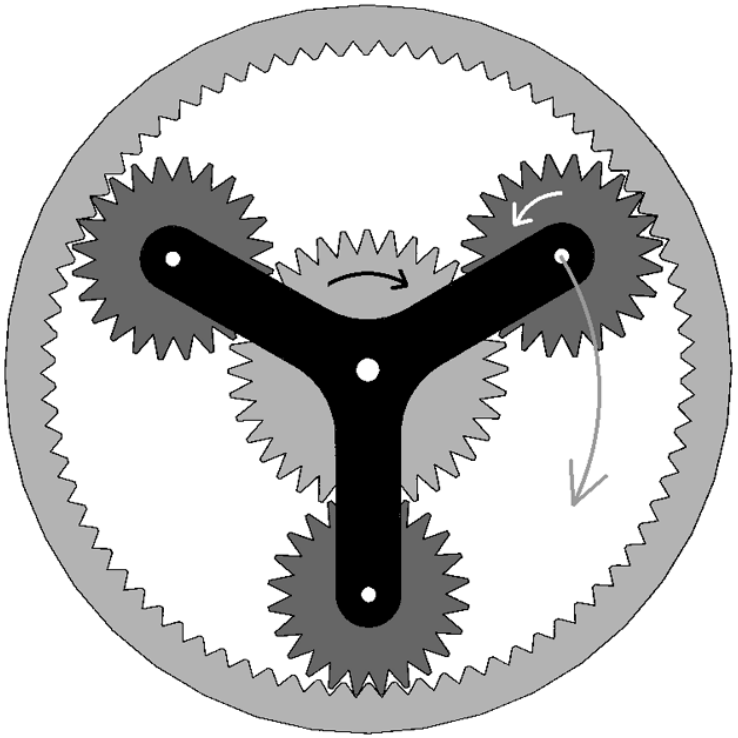
Advanced Software Testing

Integration Testing



RBCS

TIME TESTED.
TESTING IMPROVED.

www.RBCS-US.com

Advanced Software Testing

- A series of webinars, this one excerpted from *Advanced Software Testing: V3*, a book for technical test analysts, programmers, and test engineers
- People are very familiar with unit testing
- People are very familiar with system testing and unit acceptance testing
- However, integration testing is often the main missing test level
- What do software professionals need to know about integration testing?

Classic Integration Techniques

- Classic integration: functional decomposition of hierarchical system
  - Top down
  - Bottom up
  - The sandwich (a combo of top down and bottom up)
  - The big bang
- Relies on non-shippable code created by developers
  - Stubs
  - Drivers

Classic Integration Algorithm

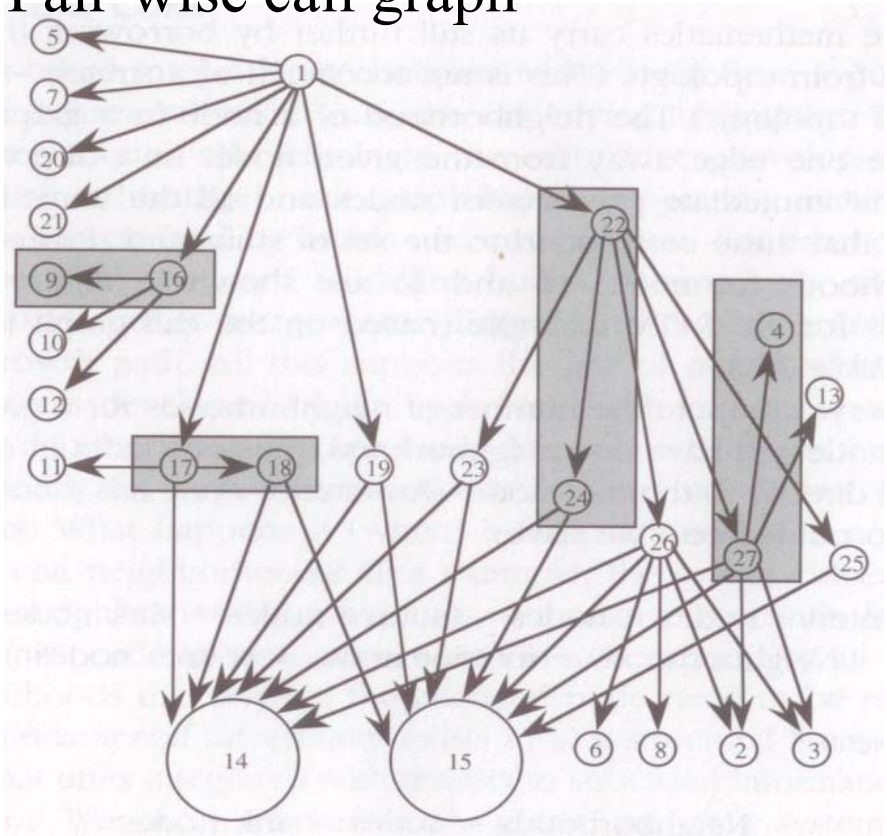
- The methodology is intuitive
  - Build incrementally with unit-tested components
  - When a failure occurs, suspect the latest-added component
- Objections to decomposition based integration:
  - Artificial workflow designed more to help project management than serve the needs of the software development team
  - Presumes that correct behavior follows from correct units and correct interfaces
  - Stubs and drivers are expensive to create
  - Excessive number of builds needed

Call Graph-Based Integration

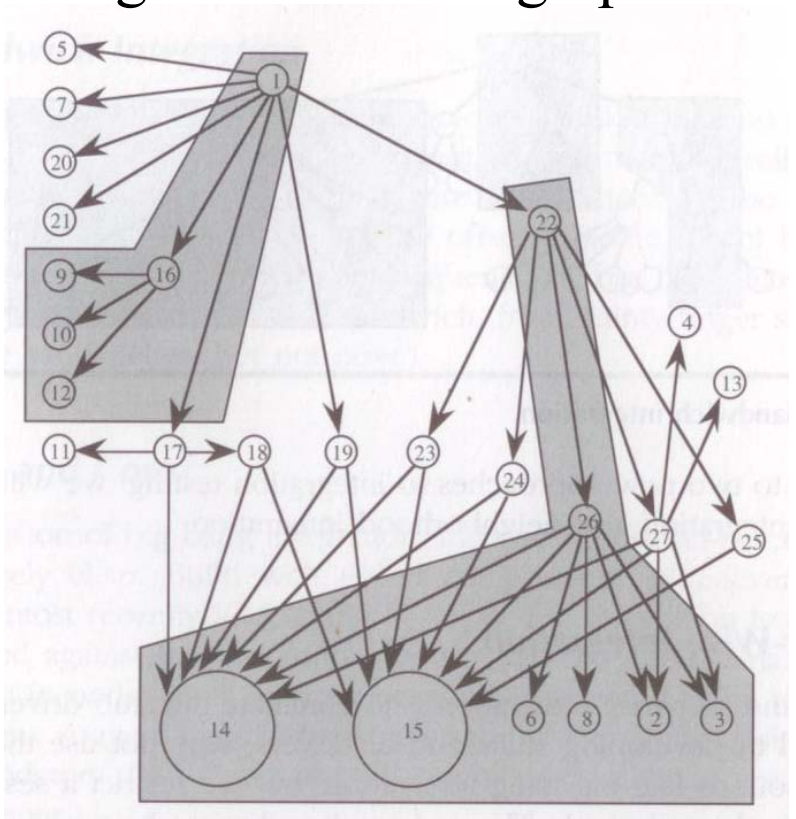
- Rather than dealing with a fixed hierarchy, this looks at the behavior of groups of modules
- Reduces non-shippable development
  - Test with real code rather than stubs and drivers
- Two basic methodologies use call-graphs
  - Pair-wise integration: does not reduce number of test sessions needed by much but does reduce stubs and drivers needed
  - Neighborhood integration: reduces both sessions and drivers and stubs

Example: Call Graph Integration

Pair-wise call graph



Neighborhood call graph

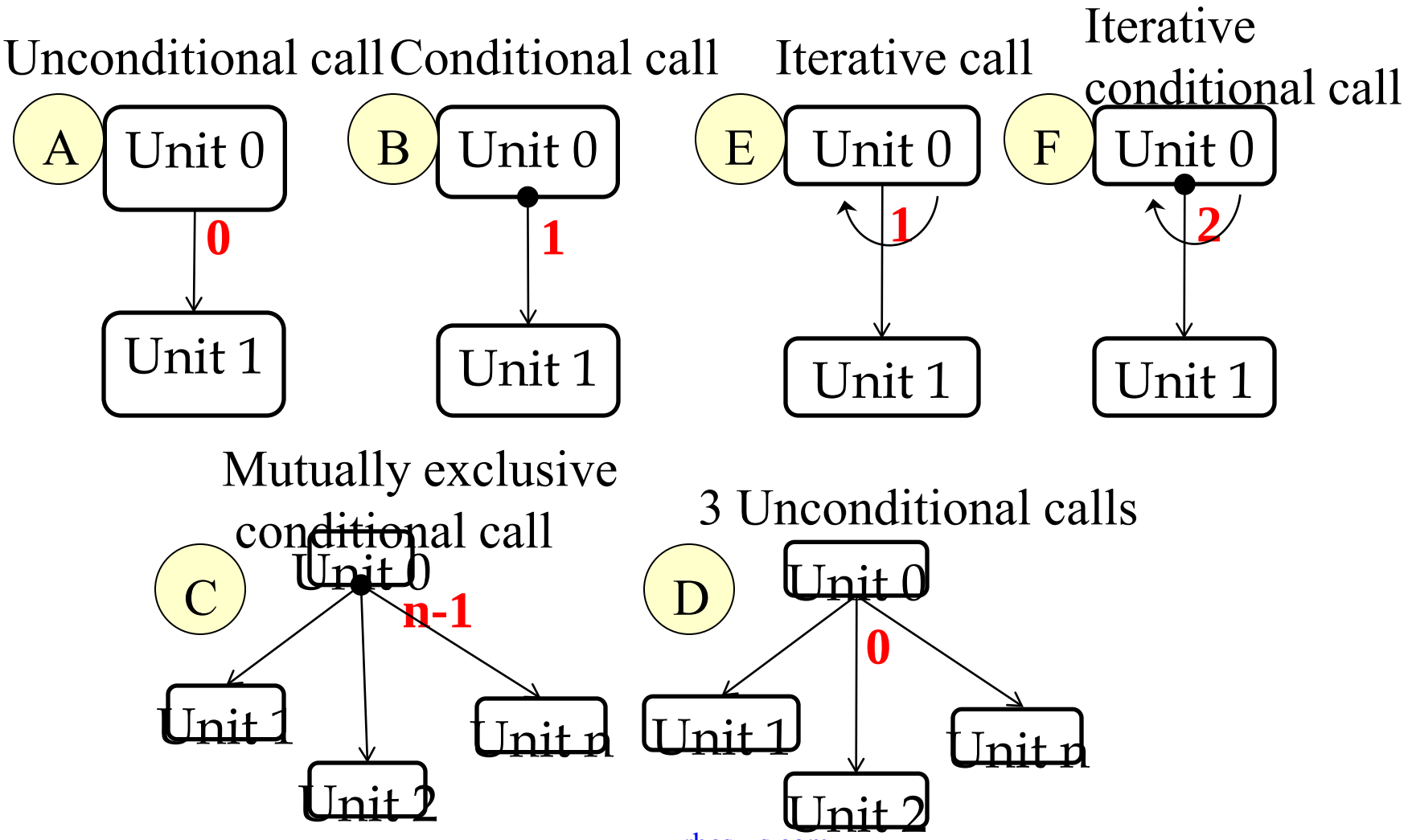


-- Graphics from "*SW Testing, A Craftsman's Approach*" by Paul C Jorgensen 2nd ed.[2002]

McCabe Integration Basis Paths

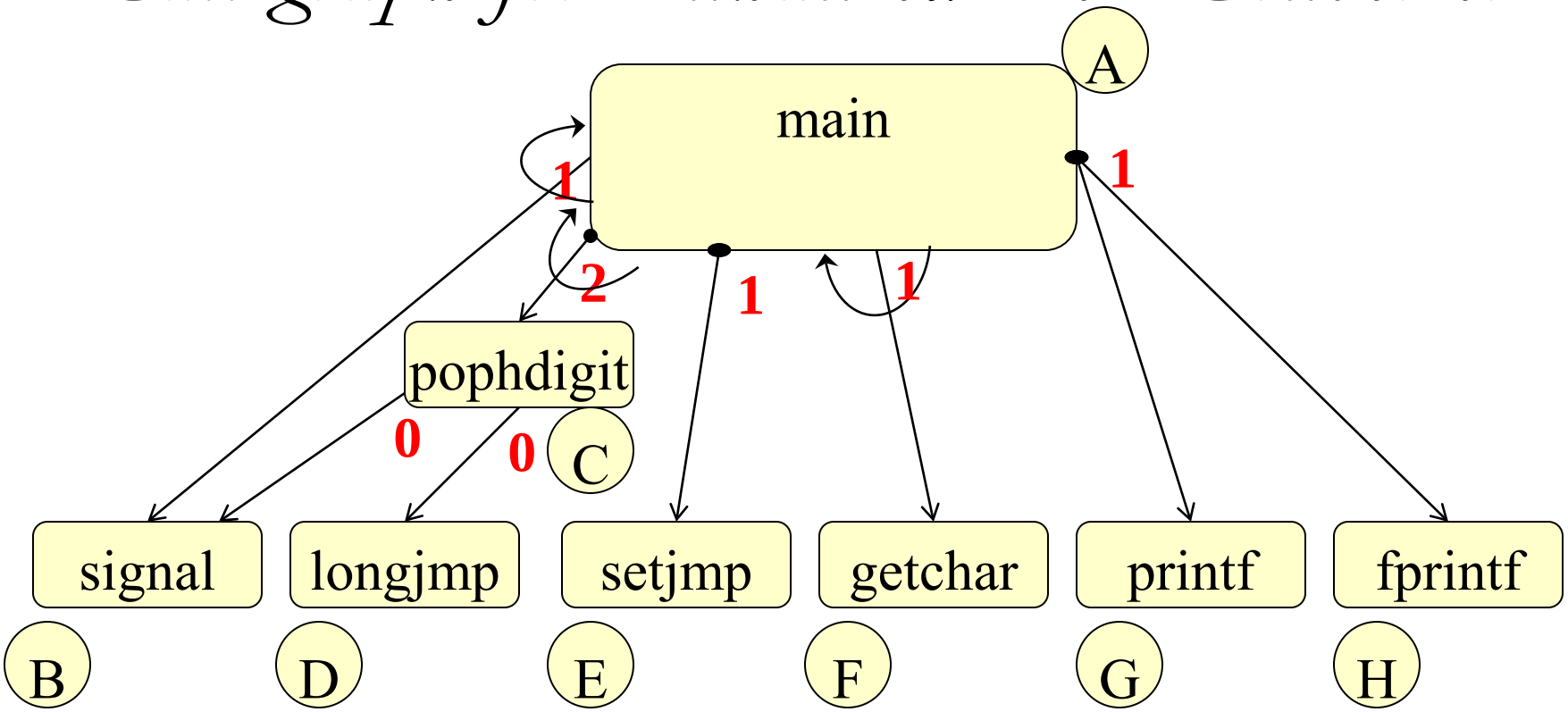
- McCabe complexity theory can also be extended and used for integration testing
- 4 different caller/callee relationships can exist between modules; any relationships in a system can be represented by one or several of these design predicates
- Each design predicate has an associated integration complexity; when combined, these indicate how many tests are needed to cover the integration effort
- Tests to cover the basis paths are called basis tests

Design Predicates



```
jmp_buf sjbuf;
unsigned long int hexnum;
unsigned long int nhex;

main()
/* Classify and count input chars */
{
  int c, gotnum;
  void pophdigit();

  hexnum = nhex = 0;

  if (signal(SIGINT, SIG_IGN) != SIG_IGN) {
    signal(SIGINT, pophdigit);
    setjmp(sjbuf);
  }
  while ((c = getchar()) != EOF) {
    switch (c) {
      case '0': case '1': case '2': case '3': case '4':
      case '5': case '6': case '7': case '8': case '9':
          /* Convert a decimal digit */
          nhex++;
          hexnum *= 0x10;
          hexnum += (c - '0');
          break;
      case 'a': case 'b': case 'c':
      case 'd': case 'e': case 'f':
          /* Convert a lower case hex digit */
          nhex++;
          hexnum *= 0x10;
          hexnum += (c - 'a' + 0xa);
          break;
      case 'A': case 'B': case 'C':
      case 'D': case 'E': case 'F':
          /* Convert an upper case hex digit */
          nhex++;
          hexnum *= 0x10;
          hexnum += (c - 'A' + 0xA);
          break;
      default:
          /* Skip any non-hex characters */
          break;
    }
  }
  if (nhex == 0) {
    fprintf(stderr, "hexcvt: no hex digits to convert!\n");
  } else {
    printf("Got %d hex digits: %x\n", nhex, hexnum);
  }

  return 0;
}

void pophdigit()
/* Pop the last hex input out of hexnum if interrupted */ {
  signal(SIGINT, pophdigit);
  hexnum /= 0x10;
  nhex--;
  longjmp(sjbuf, 0);
}
```

Call-graph for Enhanced Hex Converter



Integration Complexity = Sum(Design Predicates) + 1

IC = (**1 + 2 + 1 + 1 + 1 + 0 + 0**) + 1

IC = **7**

How Many Basis Tests are Needed?

- A basis test is a set of inputs and other conditions that force control to flow along a certain basis path
- Note that several basis paths may be covered in one test: the actual number of test cases is likely to be less than the number of basis paths
- In this case, the basis paths can be covered in 4 tests
  1. ABBEFFG (input **A**) → "**Got 1 hex digits: a**"
  2. ABFH (input **EOF**) → "**hexcvt: no hex digits to convert!**"
  3. ABBEFFFCBDFFFG (input **F5^CT9a**) → "**Got 3 hex digits: f9a**"
  4. ABBEFFCBDFF…FFCBDFFG (input a non-trivial string with lots of Ctr-C combinations) → "**Got N hex digits: XXXXXX**" where N is the number of hex digits that were loaded less the ones removed by the signals

Conclusion

- This webinar has given an overview of techniques and concepts related to integration testing
- This test level is often forgotten, but is very important
- Even in Agile lifecycles, where continuous integration occurs, tests are needed to focus on the interfaces between units
- Proper integration testing helps prevent late discovering of integration defects during system or acceptance testing

…Contact RBCS

For over a dozen years, RBCS has delivered services in consulting, outsourcing and training for software and hardware testing. Employing the industry's most experienced and recognized consultants, RBCS conducts product testing, builds and improves testing groups and hires testing staff for hundreds of clients worldwide. Ranging from Fortune 20 companies to start-ups, RBCS clients save time and money through improved product development, decreased tech support calls, improved corporate reputation and more. To learn more about RBCS, visit www.rbcs-us.com.

Address:	RBCS, Inc. 31520 Beck Road Bulverde, TX 78163-3911 USA
Phone:	+1 (830) 438-4830
Fax:	+1 (830) 438-4831
E-mail:	info@rbcs-us.com
Web:	www.rbcs-us.com