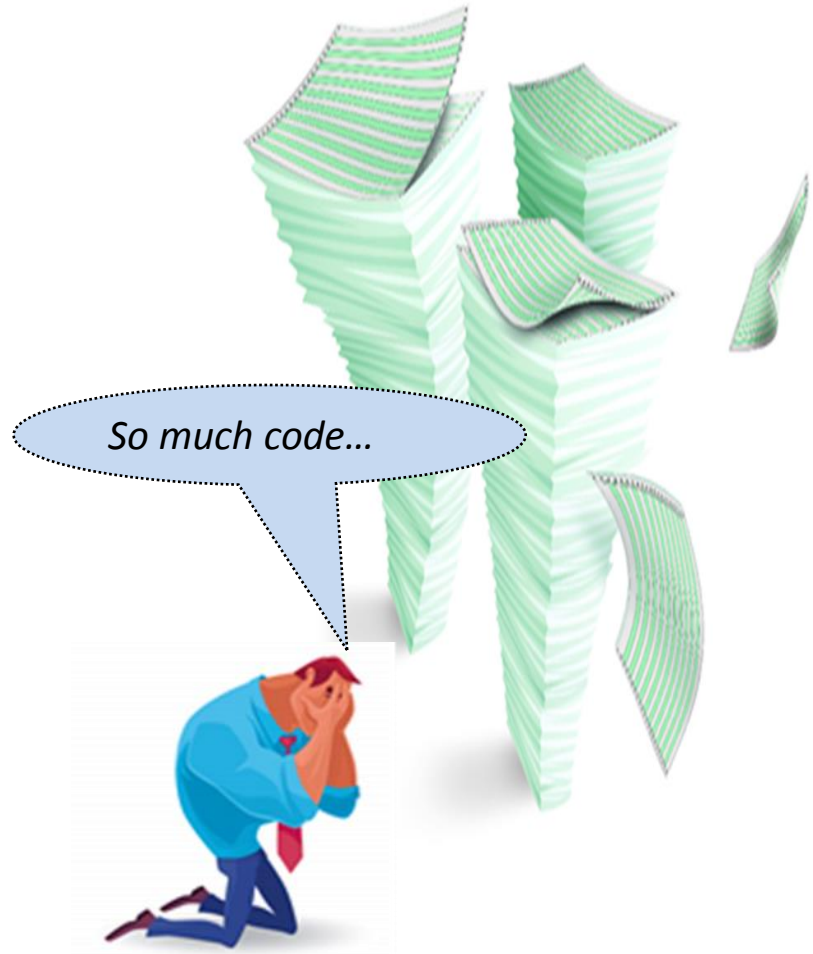
Approaches for Code Modernization

Ira Baxter Ph.D.
CTO, Semantic Designs Inc.



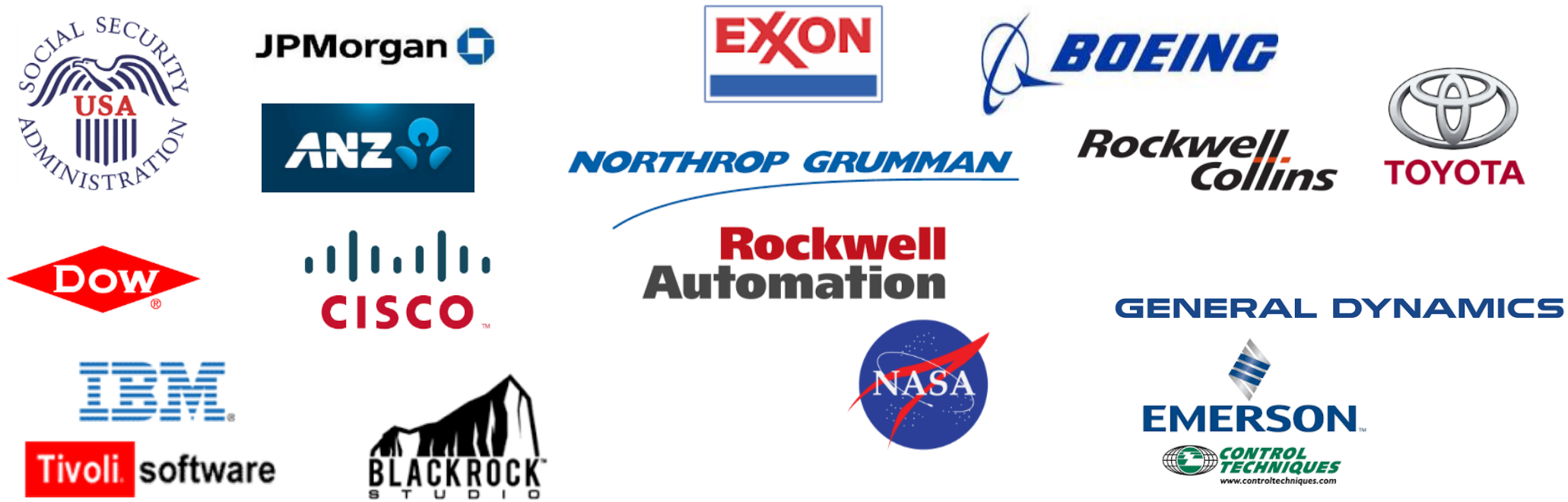
Speaker Biography: Ira Baxter

Hardcore Software Engineering Experience over 50 years

- 0th epoch: Dabbling with IBM computers in 1967: 1401, 1620, 360/40
  - Fortran, PL/1, assembler
  - Digital hardware design
- 1st epoch : 1969: Assembler and OS
  - Built my 1st Timesharing System in hardcore assembler in 1970 on 16 bit mini
  - Built/programmed 16 bit RISC minicomputer in 1973 to run 3-axis milling machines
  - Built my 2nd OS in 1975 on 8 bit micros: standalone, timeshare, distributed
- 2nd epoch : 1980: Research in foundations of automated software engineering
  - What did those operating systems have in common?
    - NOT THE ASSEMBLY CODE: ABSTRACTIONS AND DESIGN DECISIONS
  - Develop techniques to automate code transformation to find/instantiate abstractions
- 3rd epoch : 1991: Automated generation of scientific codes for super computers
- 4th epoch : 1994: R&D on software automation for factory/industrial control software
- 5th epoch : 1996: Founded Semantic Designs / Built DMS®
  - Commercial, automated software modernization services
  - Many projects essentially impossible to do manually

(*still* coding (x86) assembler support parallel symbolic computation)

- **Mission: Develop software tools to automate large-scale software analysis and change**
- Started in 1996 with $2M NIST grant for Design Maintenance System® (DMS®) concept
- Apply to software systems that exceed the capabilities of commercial COTS and vendors because of scale, complexity, and customer-specific needs.



Legacy == Successful

often mission critical

Legacy == Successful

… but …

- Code is in legacy language and/or has legacy architecture
- Data is in format that is hard to share with other systems
- Functionality is difficult to integrate with other systems
- Support costs are high
- I can’t hire new resources with legacy technology skills

➔ Response to requests for changes from clients is too long

Legacy == Successful

… but …

How to change this?

- Code is in legacy language and/or has legacy architecture
- Data is in format that is hard to share with other systems
- Functionality is difficult to integrate with other systems
- Performance is limited by legacy hardware/application structure
- Legacy engineers are retiring at accelerating rates
- Can’t hire new resources with legacy technology skills
- Support costs are high

➔ Response to requests for changes from clients takes too long

Two Basic Approaches to improving Productivity on Legacy Systems

1. Improve engineering activities

- Better processes
  - Improve specification capture to avoid implementing wrong thing
  - Improve implementation: better tools to avoid mistakes
  - Improve testing: regularize process, provide quality analysis/test generation/ tracking tools
- Educate the engineers
  - Better software engineering skills: hire or train
  - Better understanding of software structure: document architecture
  - Tools to extract useful facts from code to avoid manual discovery
  - Improve project time/cost estimations
- Recode problematic modules

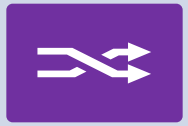
Two Basic Approaches to improving productivity on Legacy Systems

2. Modernize the code: revise *at scale* the parts that create difficulty

- Move off legacy hardware/OS where feasible
- Code in higher level languages
  - Less code/clearer structure improves engineer understanding
  - Better feedback from compilers and static analysis tools
  - Better test support
  - Less testing effort
- Better application architectures
  - More coherent subsystems
  - Less tangled code minimizes accidental interactions
  - Easier to explain to engineers
  - Can help minimize system failures
  - Better data architecture/access makes data available in broader scope

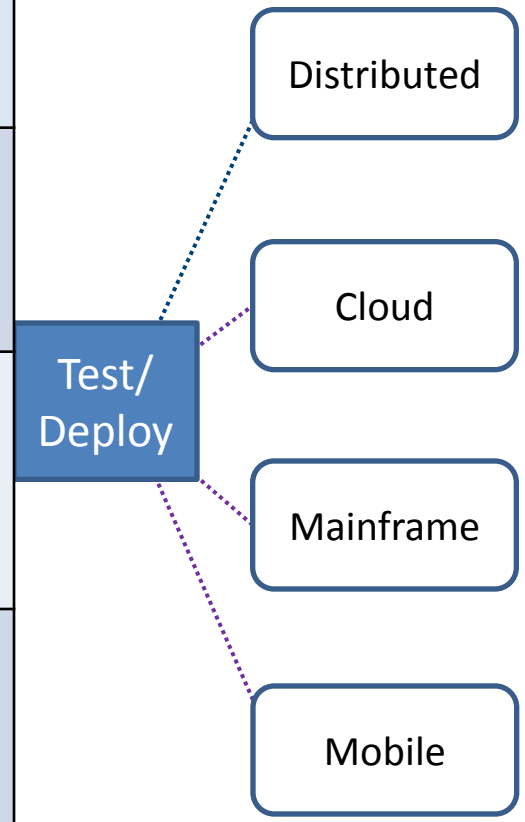
Today's focus

Modernization Strategies

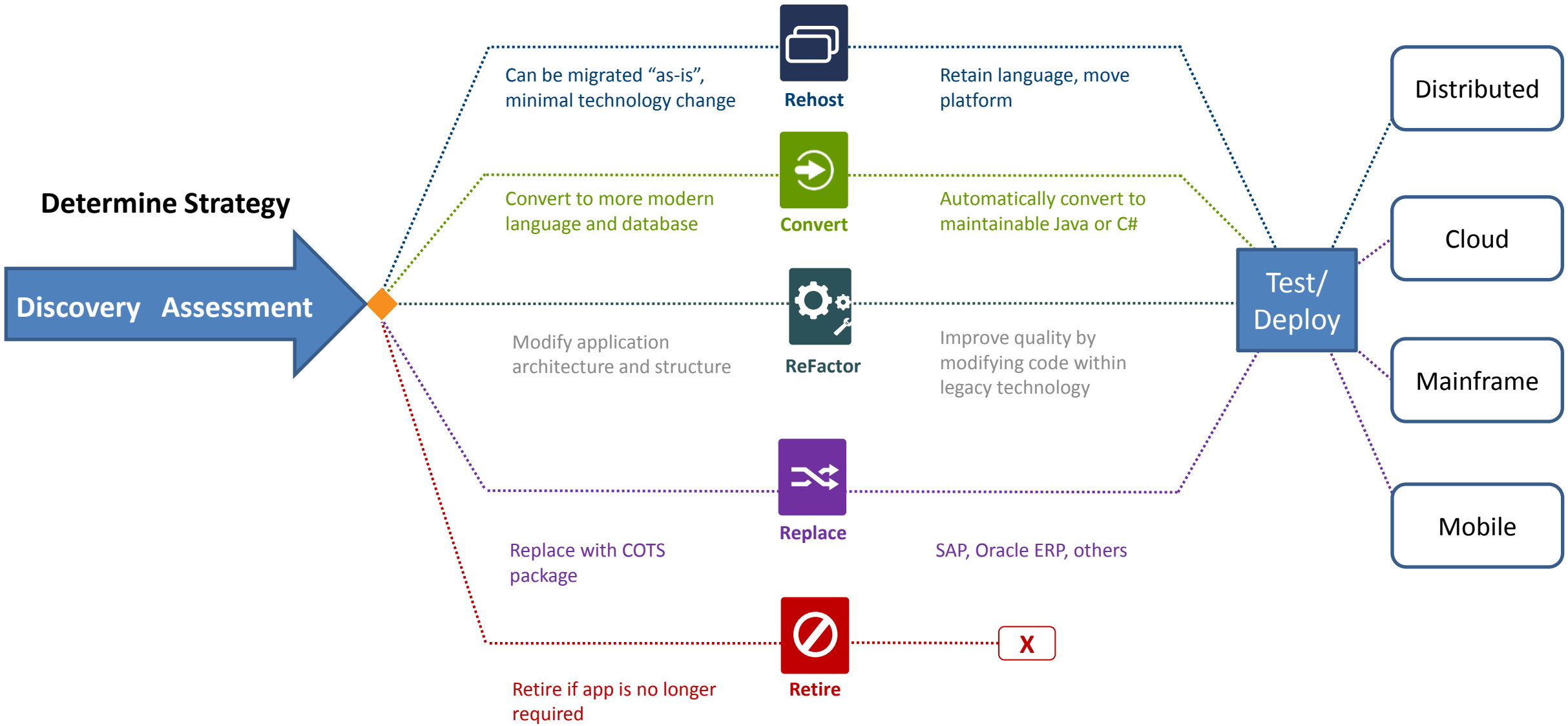
Can be migrated “as-is”, minimal technology change	 Rehost	Retain language, move platform
Convert to more modern language and database	 Convert	Automatically convert to maintainable Java/C#/C/C++
Modify application architecture and structure	 ReFactor	Improve quality by modifying code within legacy technology
Replace with COTS package	 Replace	SAP, Oracle ERP, others
Retire if app is no longer required	 Retire	

Modernization Strategies

Can be migrated “as-is”, minimal technology change	Rehost	Retain language, move platform
Convert to more modern language and database	Convert	Automatically convert to maintainable Java or C#
Modify application architecture and structure	ReFactor	Improve quality by modifying code within legacy technology
Replace with COTS package	Replace	SAP, Oracle ERP, others
Retire if app is no longer required	Retire	X

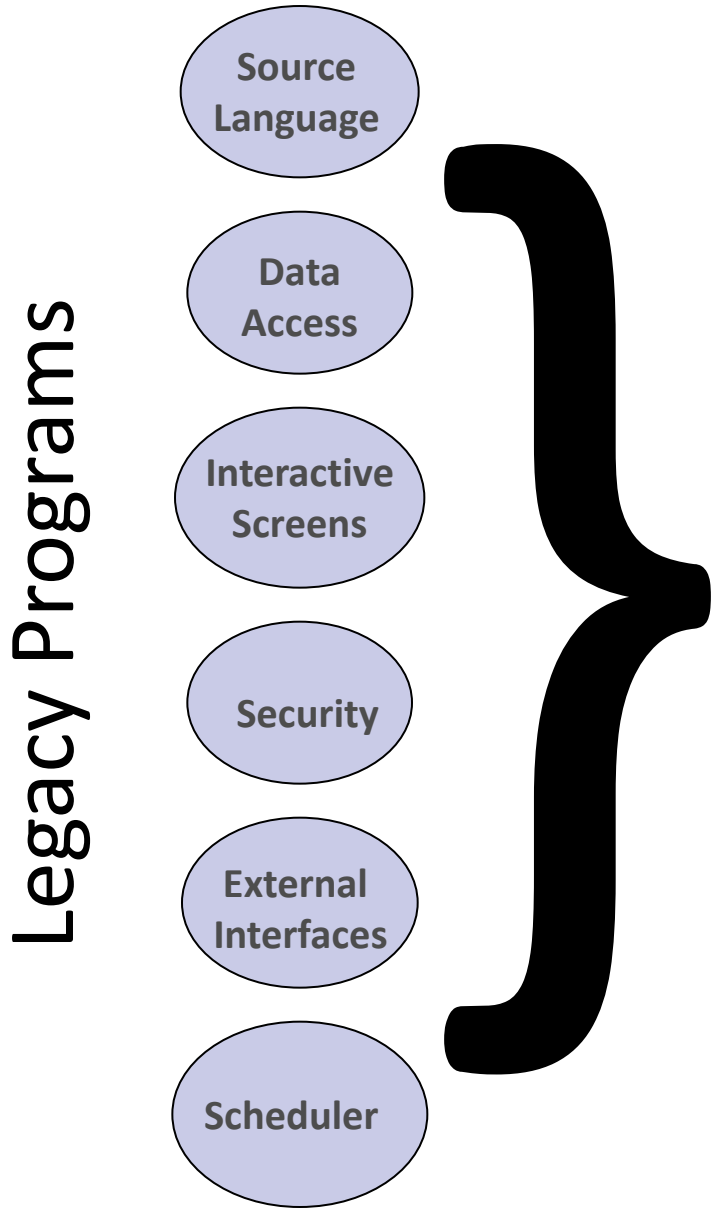


Modernization Strategies



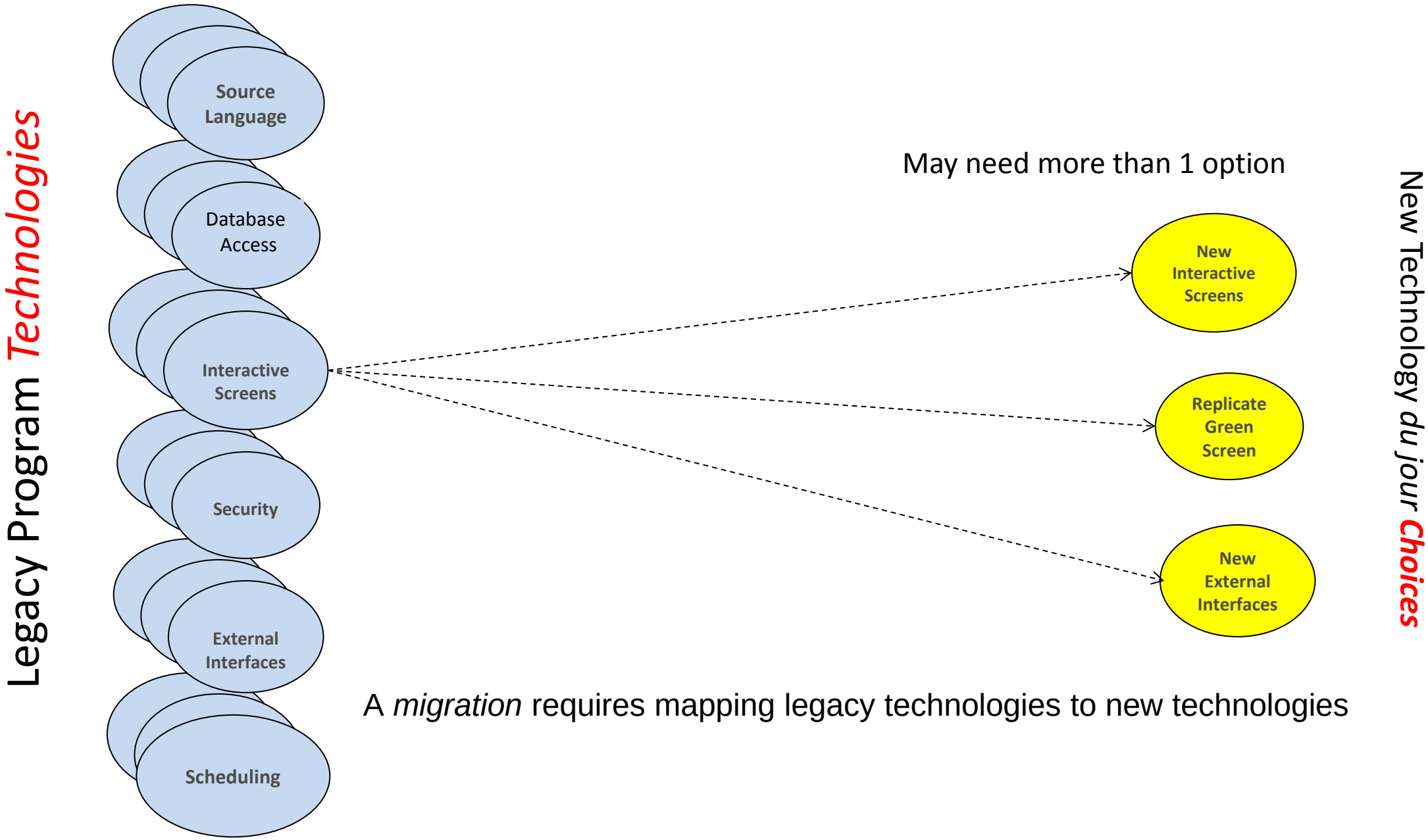
Accurate Understanding of Legacy Migration Scope

A *program* is a set of technologies *glued together* by programming language constructs



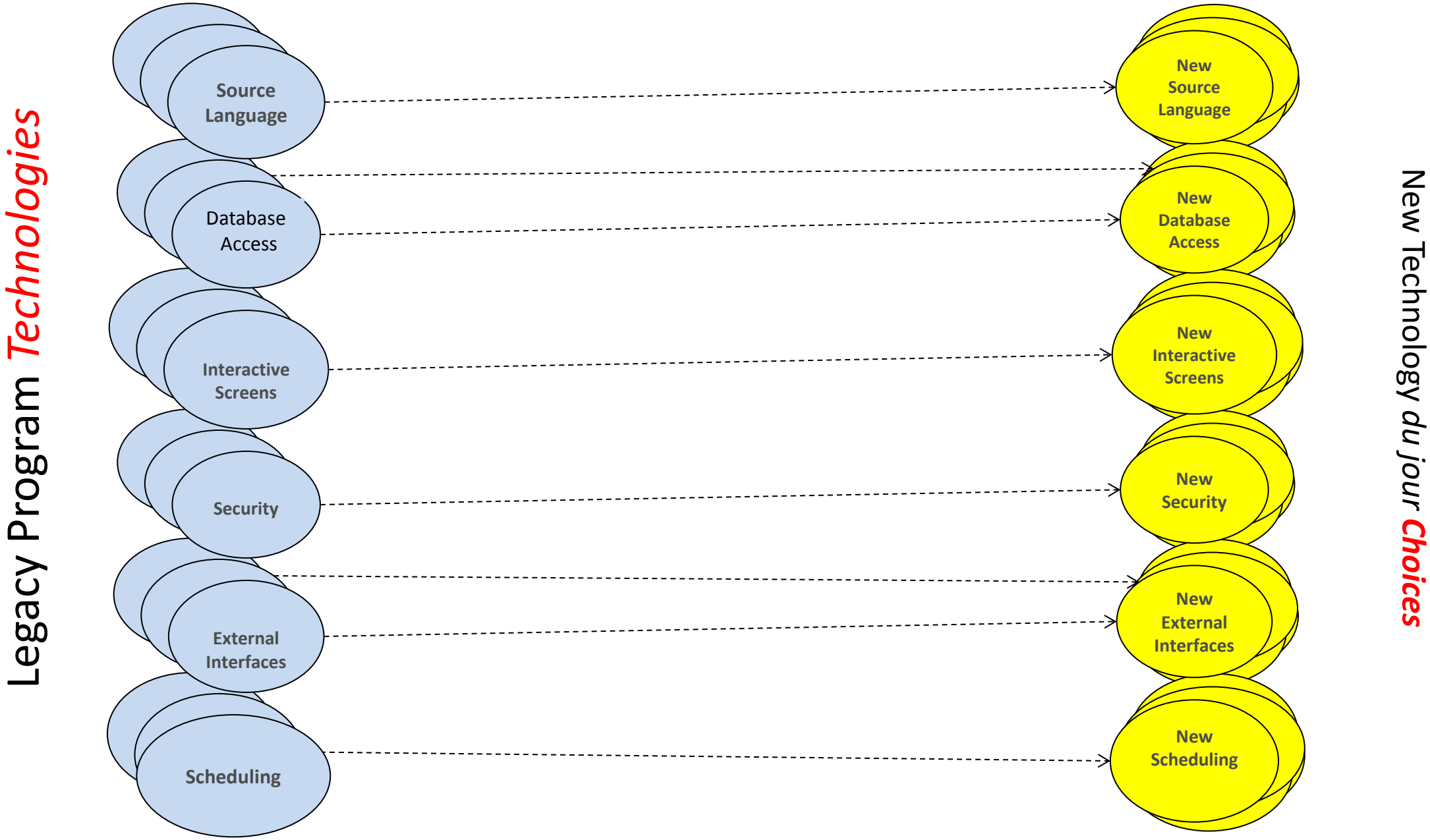
Legacy Ecosystem

Specific Technology Choices



A *migration* requires mapping legacy technologies to new technologies

Precise Strategy for each Technology Mapping



Other Considerations e.g. Code Refactoring and Optimization

Technology and Architecture Considerations

- Legacy language features where the corresponding equivalent is not readily apparent
- API conversions
- Database access issues
- Target Architecture requirements

Maintainability

- Source code format and style guidelines
- Removal of unnecessary requirements of the legacy environment
- Quality optimizations: goto removal, dead code, breaking apart monolithic structures…

Application Performance

- Translator may choose more efficient code structures (e.g, Java LONG vs PACKED)
- Modern compiler may generate more efficient code (much better code optimizers)

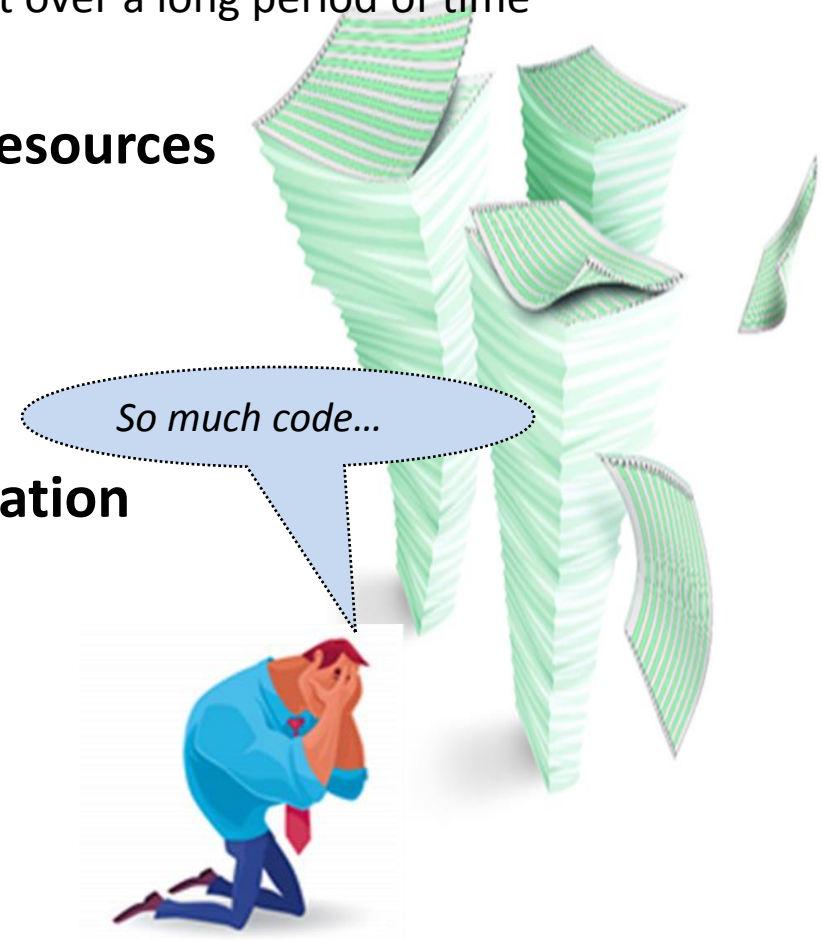
Comparison
Migration Approaches

Manual ReWrite
Point Solution Translators
Configurable Translation

Lessons Learned – Manual Migrations

The Details Matter - Huge investment in getting the legacy application right over a long period of time

- **Well known Software Development Process with skilled resources**
- **It will take longer than you think**
- **Tug of war with legacy system enhancements during migration**
- **Integration and coordination nightmare**
- **Many different coders means uneven code style/quality**
- **Humans make mistakes - Can you afford the Risk?**



Lessons Learned – Semi-Automated Migrations

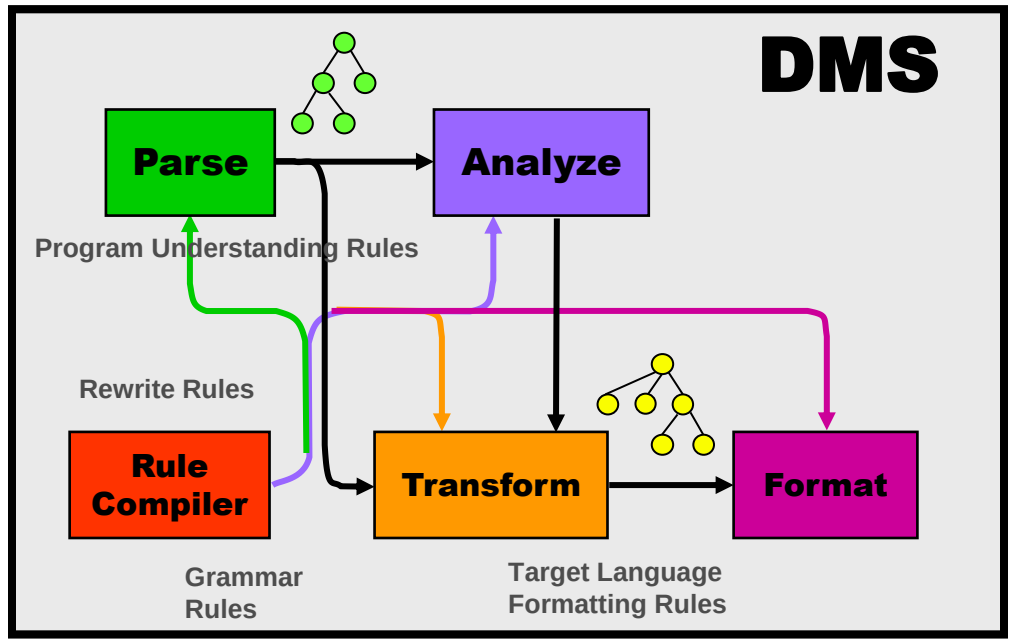
Migration Tool translated some of the code, but…

- Tremendous jump start – 50%+ translation out of the box
- Translation is incomplete - leaves hard part to do by hand
- Translation is incorrect – debugging is difficult
- Code is poor and uneven quality

Lessons Learned for Highly-Automated Migrations

Migration tool translate 99+% of the code

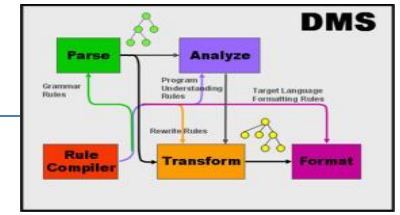
1. Translation is complete
2. Translation is correct
3. Code is consistently high quality
4. Cost 1/10 of manual migration at scale $.50 - $3.00 per LOC



Case Studies

Case Study: B2 Bomber Mission Software

NORTHROP GRUMMAN



Change: 100% Automated Migration Jovial to C

Business Challenge: Existing B-2 Mission software incapable of meeting new requirements from the military

- Legacy JOVIAL software needed to be modernized
- Internal teams unable to re-write application

Technical Problem: Legacy Software Complexity

- Client tried twice and failed before turning to Semantic Designs
- 1.2 million lines Black code; ***SD not allowed to see source***

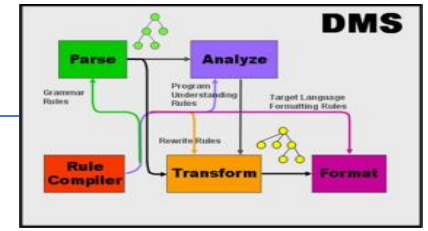
Solution: *Migrated 100% by DMS*

- Define JOVIAL language from scratch to DMS
- Reuse existing definition for C target language
- ~6000 translation rules
- Delivered in 9 months

•**Benefit: Trustworthy solution for critical software**

Jovial to C conversion
Operational in US Strategic B2 Bomber fleet

Case Study: Flight Reservations Migration Airlines



Change: Convert legacy programing language to C

Business Challenge: Programmers creating new system defects when making application changes

- Old technology with unmaintained compiler on TPF
- Aging workforce unable to hire resources with SabreTalk skills
- Speed to market for system modifications

Technical Problem: Scale makes manual conversion to risky

- 5+ Million lines of SabreTalk plus Assembler Macros
- 14,000 software components

Solution: 90 Day Proof of Concept Migration

- 11 key modules 100% automatically translated by DMS Toolkit
- Validated by Customer - Transitioning into production
- Demo conversion of remaining modules at 95% automated conversion

Benefit: Lower Cost and Risk for Migration

They also want help:

- Understanding programs
- Testing migration
- Testing modified code

```
OPTIONS=TRACE
 FINDCUST: PROC;
   /* TINY SABRETALK PROGRAM */
   %INCLUDE EB0EB,SW00SR,INVDB,INVCES;
   DCL FOUND BIT(32);
   DCL CUSTNAME CHAR(64) BASED(CUSTPTR);
   DCL BUFFER CHAR(100) BASED(BUFPTR);
   DCL INVPTR BIT(32);
   DCL DEADVAR BIT(32);
   DCL SUB BIN(15) DEF EBW096;
   /*************************/
   START (CUSTPTR=#RG1);
   INVDBPTR = INVDBCR2;
   BSTR(BUFFER,1,1) = 0;
   CSTR(BUFFER,2,99) = CSTR(BUFFER,1,99);
   BSTR(EBSW01) = '00'X;
   INVPTR = INVDBA90(SUB);
 FNDEAD: FOUND=0;
   IF INVPTR = 0
      THEN GOTO FNDC45;
   GOTO FNDOPEN;
 FNDREAD: DBRED 'REF=INVCES','REG=R1',
       KEY1=(PKY=#INVCESK),
       UP,BEGIN,ERRORA=FNDC45
       (#RG1=INVPTR,#RG3=SW0PTR,
       SW0PTR=#RG3,INVPTR=#RG1);
   IF SW00RTN > 0
      THEN GOTO FNDC45;
   CALL SETFOUND;
 FNDC45: DBCLS 'REF=INVCES',RELFC
           (SW0PTR=#RG3);
 EXIT:
   BACKC(#RG1=BUFPTR);
 FNDOPEN: DBOPN 'REF=INVCES','REG=R1',
            FADDR=?INVPTR,HOLD
            (SW0PTR=#RG3);
 GOTO FNDREAD;

 SETFOUND: PROC;
   FOUND = 1;
 END SETFOUND;

 DEADPROC: PROC;
   FOUND = 0;
 END DEADPROC;

 END FINDCUST;
```

Example: SabreTalk Automatically translated to C

Some C translator features:

- *Main program*
- *Subroutines*
- *Header files*
- *GOTOs removed*
- *Dead code/labels/data removed*
- *Optimized clears/assignments*
- *… many more*

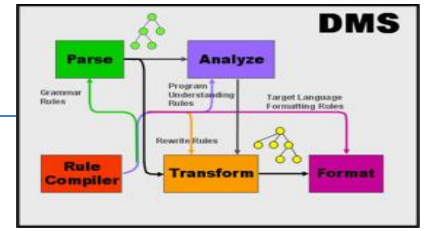
```
/* OPTIONS=TRACE */
#include …
/* TINY SABRETALK PROGRAM */
#include <tpf/c_eb0eb.h>
#include <c_sw00sr.h>
#include <invdb.h>
#include <invces.h>
static void setfound(ULong4_t * const found);

void findcust(struct TPF_regs *regs_)
{ regs_->r1 =  (long int)findcust_(
                       (char (*)[64])regs_->r1);
}

char (*findcust_(char (*custptr)[64]))[100]
{ struct sw00sr *sw0ptr = NULL;
  INVDB_t *invdbptr = NULL;
  INVREC_t *invptr = NULL;
  ULong4_t found = 0x0;
  char (*bufptr)[100] = NULL;
  ULong4_t invptr = 0x0;
  Short2_t * const sub =
                  (Short2_t *)&ecbptr()->ebw096;
  /*************************/
  invdbptr = (INVDB_t *)ecbptr()->invdbuf;
  *(UChar_t *)bufptr &= ~ 0x80;
  memcpy(& ((char *)bufptr)[1], bufptr, 99);
  ecbptr()->ebsw01 = 0x0;
  invptr = invdbptr->invreca[*sub - 1].invb70;
  found = 0;
  if (invptr != 0)
    { sw0ptr = dfopn_acc("INVCES", INVCES_ID,
                DFOPN_FADDR, DFOPN_HOLD, invptr);
      dft_kyl keys_;
      memset(&keys_, 0, sizeof keys_);
      df_setkey(&keys_, 1,
         offsetof(invdbrec_t, invcesk),
         member_size(invdbrec_t, invcesk),
         DF_EQ, 0, invdbrec, DF_UPORG, DF_CONST);
      dfkey_nbr(sw0ptr, &keys_, 1);
      invptr = (INVREC_t *)dfred(sw0ptr,
                              DFRED_BEGIN);
      if (! DF_ER(sw0ptr)
          && sw0ptr->sw00rtn == 0)
        { setfound(&found);
        }
    }
  sw0ptr = dfifb_ref("INVCES");
  dfcls(sw0ptr, DFCLS_RELFC);
  return bufptr;
}

static void setfound(ULong4_t * const found)
{ *found = 1;
}
```

Case Study: Mainframe Data Processing



Change: Convert HLASM to C

Business Challenge: Legacy HLASM critical to large-scale payroll delivery

- z/OS and OS/MFT simulation modules
- Very small remaining pool of HLASM engineers
- Significant risk to business continuity

payroll processing

Technical Problem: Manual conversion of HLASM is *hard*

- 250K SLOC z/OS and OS/MFT (simulation) + Macros
- HLASM engineers have other full time duties

Solution: 90 Day Proof of Concept Migration

- Client-chosen modules 95% automatically translated by DMS Toolkit

Benefit: Lower Cost and Risk for Migration

Considering HLASM to COBOL

```
*                                                                       02420000
*    MAJOR FORMAT CODE 2: EUROPEAN (DDMMYYYY)                           02430000
*                                                                       02440000
MFMT2    BAL   R9,MFMTSUB              GO GET SEPARATOR CHARACTOR       02450000
         LA    R2,POUTDATE             LOAD OUTPUT AREA ADDRESS         02460000
         MVC   0(2,R2),PDATE+2         MOVE DD                          02470000
         LA    R2,2(,R2)               BUMP OUTPUT ADDRESS              02480000
         LTR   R3,R3                   IS THERE AN INSERTION CHAR?      02490000
         BZ    MFMT2A                  NO, JUMP OVER                    02500000
         STC   R3,0(,R2)               INSERT CHARACTER                 02510000
         LA    R2,1(,R2)               BUMP OUTPUT ADDRESS              02520000
MFMT2A   MVC   0(2,R2),PDATE           MOVE MM                          02530000
         LA    R2,2(,R2)               BUMP OUTPUT ADDRESS              02540000
         LTR   R3,R3                   IS THERE AN INSERTION CHAR?      02550000
         BZ    MFMT2B                  NO, JUMP OVER                    02560000
         STC   R3,0(,R2)               INSERT CHARACTER                 02570000
         LA    R2,1(,R2)               BUMP OUTPUT ADDRESS              02580000
MFMT2B   MVC   0(4,R2),PDATE+4         MOVE YYYY                        02590000
         LA    R2,4(,R2)               BUMP OUTPUT ADDRESS              02600000
         B     SETSIZE                 SET OUTPUT FIELD SIZE            02610000
*                                                                       02620000
*    MAJOR FORMAT CODE 3: F.I.P.S. (YYYYMMDD)                           02630000
*                                                                       02640000
MFMT3    BAL   R9,MFMTSUB              GO GET SEPARATOR CHARACTOR       02650000
         LA    R2,POUTDATE             LOAD OUTPUT AREA ADDRESS         02660000
         MVC   0(4,R2),PDATE+4         MOVE YYYY                        02670000
         LA    R2,4(,R2)               BUMP OUTPUT ADDRESS              02680000
         LTR   R3,R3                   IS THERE AN INSERTION CHAR?      02690000
         BZ    MFMT3A                  NO, JUMP OVER                    02700000
         STC   R3,0(,R2)               INSERT CHARACTER                 02710000
         LA    R2,1(,R2)               BUMP OUTPUT ADDRESS              02720000
MFMT3A   MVC   0(2,R2),PDATE           MOVE MM                          02730000
         LA    R2,2(,R2)               BUMP OUTPUT ADDRESS              02740000
         LTR   R3,R3                   IS THERE AN INSERTION CHAR?      02750000
         BZ    MFMT3B                  NO, JUMP OVER                    02760000
         STC   R3,0(,R2)               INSERT CHARACTER                 02770000
         LA    R2,1(,R2)               BUMP OUTPUT ADDRESS              02780000
MFMT3B   MVC   0(2,R2),PDATE+2         MOVE DD                          02790000
         LA    R2,2(,R2)               BUMP OUTPUT ADDRESS              02800000
         B     SETSIZE                 SET OUTPUT FIELD SIZE            02810000
*                                                                       02820000
*    THIS SUBROUTINE PLACES THE INSERTION CHARACTER (IF ANY) INTO       02830000
*    REGISTER 3 BASED UPON MINOR CODE 1 THROUGH 5 (USED BY MAJOR        02840000
*    CODE ROUTINES 1 THROUGH 3 ABOVE.                                   02850000
*                                                                       02860000
MFMTSUB  XR    R3,R3                  ASSUME NO SEPARATOR              02870000
         XR    R4,R4                  CLEAR WORK REGISTER              02880000
         IC    R4,PMINOR              GET MINOR FORMAT NUMBER          02890000
         N     R4,=F'15'              CLEAR SIGN                       02900000
         BCTR  R4,0                   DECREMENT BY 1 (ZERO BASED)      02910000
         SLL   R4,2                   MULTIPLY BY 4 (LENGTH OF BRANCH) 02920000
         LA    R4,MFMTSBR(R4)         LOAD ADDRESS OF BRANCH           02930000
         BR    R4                     GO BRANCH DEPENDING ON FORMAT    02940000
*                                                                       02950000
*    BRANCH TABLE TO SELECT INSERTION CHARACTER                         02960000
*                                                                       02970000
MFMTSBR  B     MFMTSC1                NONE                             02980000
         B     MFMTSC2                SPACE                            02990000
         B     MFMTSC3                SLASH                            03000000
         B     MFMTSC4                HYPHEN                           03010000
         B     MFMTSC5                PERIOD                           03020000
```

Example: HLASM Automatically translated to C

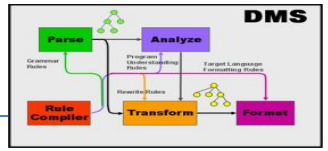
C translator features:

- *All assembly artifacts gone (registers, CC)*
- *Fully structured (goto-free) code*
- *Discovery/formation of subroutines w/ parameters*
- *JMP table to switch statement*
- *Conversion of DSECT to structs*
- *Discovery of arrays*
- *… many more*

```
...

case 2:
  {
    // *
    // *    major format code 2: european (ddmmyyyy)
    // *
    // label: mfmt2
    fnMfmtsub(&c, _parms); // go get separator charactor
    (&_parms->Poutdate)[0] = _parms->Pdate[2],
    (&_parms->Poutdate)[1] = _parms->Pdate[3]; // load output area address // move dd
    pc = &_parms->Poutdate + 2; // load output area address // bump output address
    if (c != 0) { // is there an insertion char?
      *pc++ = c; // insert character
    }
    // label: mfmt2a
    *pc++ = _parms->Pdate[0], *pc++ = _parms->Pdate[1]; // move mm
    if (c != 0) { // is there an insertion char?
      *pc++ = c; // insert character
    }
    // label: mfmt2b
    *pc++ = _parms->Pdate[4], *pc++ = _parms->Pdate[5],
    *pc++ = _parms->Pdate[6], *pc++ = _parms->Pdate[7]; // move yyyy
    break;
  }
case 3:
  {
    // *
    // *    major format code 3: f.i.p.s. (yyyymmdd)
    // *
    // label: mfmt3
    fnMfmtsub(&c, _parms); // go get separator charactor
    (&_parms->Poutdate)[0] = _parms->Pdate[4],
    (&_parms->Poutdate)[1] = _parms->Pdate[5],
    (&_parms->Poutdate)[2] = _parms->Pdate[6],
    (&_parms->Poutdate)[3] = _parms->Pdate[7]; // load output area address // move yyyy
    pc = &_parms->Poutdate + 4; // load output area address // bump output address
    if (c != 0) { // is there an insertion char?
      *pc++ = c; // insert character
    }
    // label: mfmt3a
    *pc++ = _parms->Pdate[0], *pc++ = _parms->Pdate[1]; // move mm
    if (c != 0) { // is there an insertion char?
      *pc++ = c; // insert character
    }
    // label: mfmt3b
    *pc++ = _parms->Pdate[2], *pc++ = _parms->Pdate[3]; // move dd
    break;
  }
```

Case Study: Automated System Refactoring



***Change*: Modify System to guarantee quality of service for critical aircraft functions**

Business Challenge: Add management for real time video data

- Product line used in several military airframes

Technical Problem: 6,000 C++ Modules needed refactoring

- Avionics Mission Software ~ 5M SLOC
- Architected in 1992 as components with monolithic API's
- Replace APIs for Boeing custom OS everywhere

Solution: 100% Refactoring with DMS

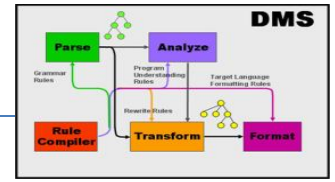
- Re-architect code into more reusable parts
- Restructure APIs into conceptually clean groups
- Move towards CORBA/RT component model
- Change communication to use ORBs

Multiple UAVs finding/targeting tactical targets

***Live Fire*: F-16+JDAM and HIMARS+ATACMS**

Benefit: Trustworthy solution for critical software at scale

Case Study: Chemical Plants



Change: Model/Migrate Software Running Manufacturing Process

Business Challenge: Trusted plant-controller computers starting to fail due to age

- Many different plants / **Thousands of control programs**
- Software had to be migrate to modern controller hardware
- Limited resources and time

Some plants now converted

Technical Challenge: Manual conversion impractical for scale

- Can't be wrong or factory may "blow up"
- Assembly like language difficult to analyze

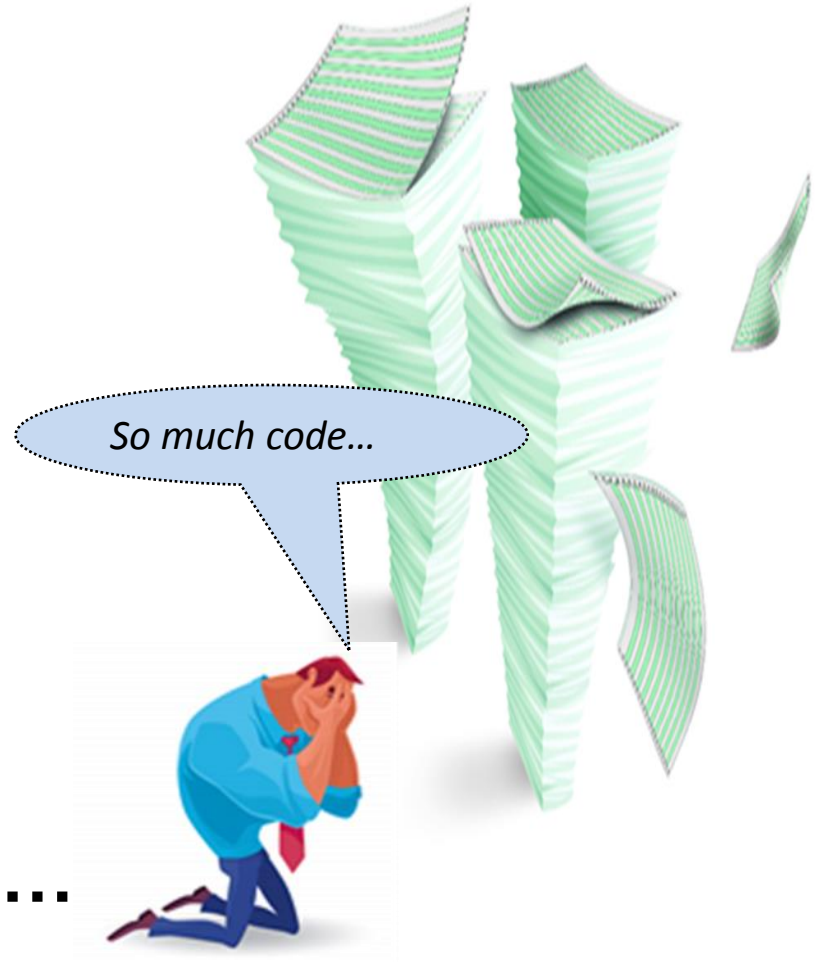
Solution: Automated Tool to recover abstract process control model from "assembly code"

- Define Dowtran from scratch to DMS
- **Define abstractions in terms of data flows** with conditional implementations
- **DMS matches legacy code via data flows** ("Programmer's Apprentice") to produce model
- Generate new controller code from model

Benefit: Reliable migration of business/safety critical software + huge cost savings + *design capture*

Questions?

idbaxter@semanticdesigns.com



… at Vendor Hospitality later today…

… at TPF conference thru Wednesday …