ERP LANGUAGE CHALLENGES

Roman Ivantsov, MCSD
Software Architect, Tyler Technologies, Eden Division
www.tylertech.com
roman.ivantsov@tylertech.com

ERP – Enterprise Resource Planning - hardware or software system that serves all departments within an enterprise

Basic Facts:

- Code size: millions of code lines
- Number of users: tens to thousands
- Cost: millions to hundreds of millions $US
- Launch period: years
- Service time: 10+ years

Challenge: No two businesses are the same

Extended Configuration (setup) and Customization (coding)

Facts:

- SAP ERP has 3000 configuration tables
- Licensing fee is only 30% of total cost

ERP implementation is extremely complex, costly and risky enterprise

Sad Facts:

- US Navy ERP project - $1 bln wasted
- Hershey's 112 million ERP disaster
- Fox-Meyer bankruptcy from ERP disaster – vendors were sued for $500 mln each

Causes of failures

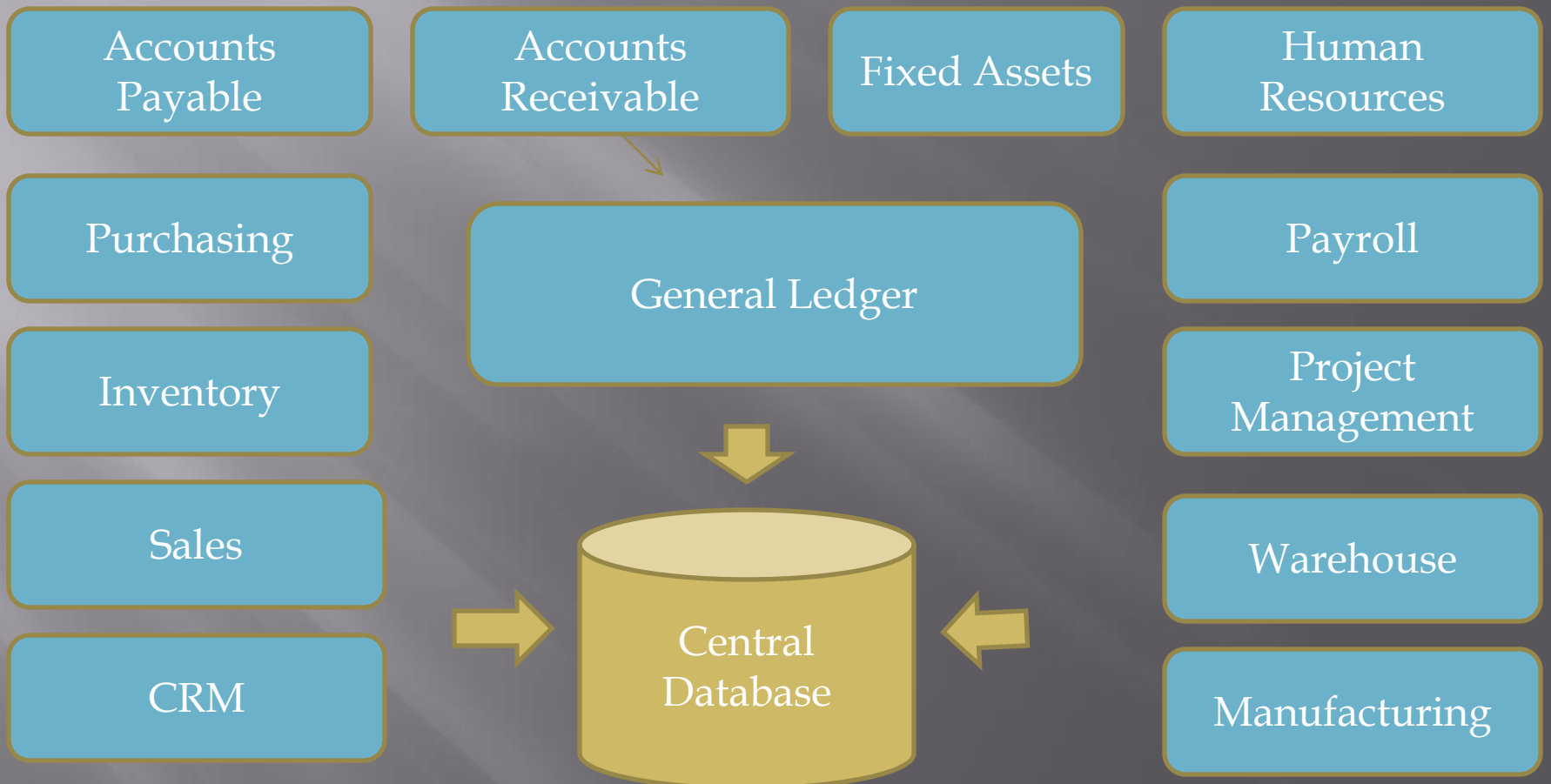
Multiple things are blamed for failures: technical, financial, project management, …

I suggest

Inadequate development language and environment

ERP needs more attention from language research community. It manifests extremes for problems that are common to other areas. ERP's huge economic impact more than justifies this attention.

ERP Structure



Multi-site distributed development model

Software vendor ships bits with full source code

ISVs – localization, custom modules, country-specific functionality

On-site programming team

ERP Language Environment

- Business logic is coded in a special business-oriented language: ABAP, Informix-4GL, X++, Cobol
- Procedural and OO
- Built-in data access support (like LINQ)
- Multi-site development - mostly by code overwrite at file level

What can be done better?

New business-oriented language

- Openness – better ways to override/overwrite code
- Long-running processes (persistent continuations)
- Advanced object model: data binding, authorization, transactions, indestructible objects
- Built-in messaging (Erlang)
- Inter-module interfaces: scripting, not API
- Error-handling – (Scheme conditions)
- Versioning and customization
- Testing and mock data
- Localization, schema changes sync, UI construction

Code openness

- Partial classes/modules – with parts in different assemblies
- Extensible enumerations (and switch statements)
- CLOS-like multiple-type dispatch - by argument types and enumeration values
- “new” operator override – built-in dependency injection
- Private/public modifiers are less important – just recommendations

Processes and communications

- Persistent continuations instead of Workflow modeling
- Built-in messaging (Erlang's *receive*)
- Light-weight non-OS threads; easy parallel execution and locking
- Error handling – ability to recover failed process (Scheme's conditions)

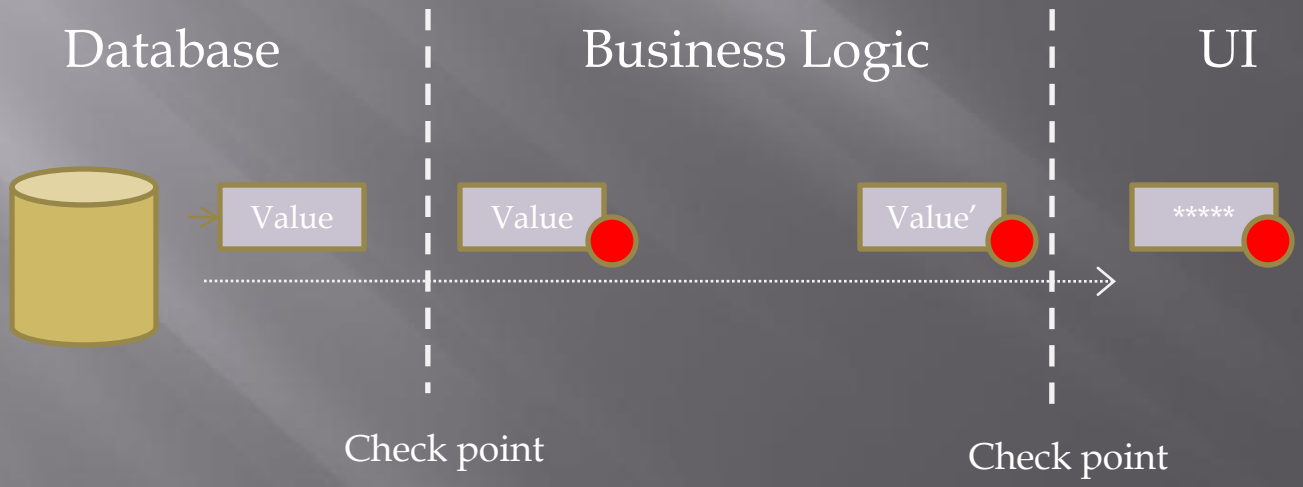
Object Model

- True message passing - method call is a message that can be intercepted (AOP style)
- Automatic change events for properties/fields
- Data-binding – everywhere, not only in UI. Calculated fields
- Transaction support
- Lazy evaluation
- Indestructible virtual objects – “obligation”

Authorization Framework

Every data value has attached "authorization token" (similar to Tainted flag in Ruby). Runtime creates/checks the token each time the value crosses authorization zone and clears the value if necessary.

Research group at UC proposed similar model for security in JVM.



Public module interfaces

Script instead of API (imagine the world without SQL?!)

Back to McCarthy's Business Communication Language idea (1969) – Lisp-like data format and query language in one

Other features

- Versioning and customization (conditionals are not enough)
- Testing – ability to test stand-alone method without full compile; mock screens and test data
- Localization support
- Schema changes synchronization
- UI awareness, automatic UI construction

Raise the Abstraction Level

- Remove “abstraction leaks” – prevent quick code aging
- Support embedded DSLs (macros and custom language constructs) – to make “code” readable by business people
- The dream goal – join specs and code in one document
- Making changes must be easier – break the deadly spec-to-code loop

Focus Change

From

Productivity

To

Maintainability and Extensibility

Because every useful code lives too long!

Conclusions

- ERP systems manifest extreme cases in almost all aspects of software development
- A new language environment is urgently needed to meet new ERP challenges
- ERP economic impact – and possible payoff – is significant enough to justify extended research efforts.