1. You Oversell Agents

Agents are not magic! If you can’t do it with ordinary software, you probably can’t do it with agents. No evidence that any system developed using agent technology could not have been built just as easily using non-agent technology if you can’t do it with ordinary software, you probably can’t do it with agents. Agents are not AI by a back door. Don’t equate agents and AI.

2. You Get Religious

Agents have been used in a wide range of applications, but they are not a universal solution. For many applications, conventional software paradigms (e.g., OO) are more appropriate. Given a problem for which an agent and a non-agent approach appear equally good, prefer non-agent solution. Agents may make it easier to solve certain classes of problems better, but they do not make the impossible possible. Other form of dogma: believing in your agent definition.

1. Pitfalls of Agent Development

- Implementation
- Macro (society) level
- Micro (agent) level
- Analysis and design
- Conceptual
- Methodology
- Poltical

Seven categories: Identifies key pitfalls.

We now consider pragmatics of Ao software projects. Development needs more attention. Lots of (single and multi-agent) software projects... but agent-oriented...
1.3 Don't Know Why You Want Agents

Agents = new technology = lotsofhype!

"Agents will generate US$2.6 billion in revenue by the year 2000."

Managerial reaction:
"We can get 10% of that."

Managers often propose agent projects without having clear ideas about what, where to go with them. The lesson: understand your reasons for attempting an agent, development project, and what you expect to gain from it.

No business plan for the project:

No technology vendor?

No solutions vendor?

No pure research?

The lesson: be sure you understand how and where your new technology may be most usefully applied. Puts the cart before the horse.

Having developed some agent technology, you search for an application to use them.

The ‘yet another agent testbed’ syndrome.

1.4 Don't Know What Agents Are Good For

Having developed some agent technology, you search for an application to use them.

Putting the cart before the horse.

The ‘yet another agent testbed’ syndrome.

Putting the cart before the horse.

The ‘yet another agent testbed’ syndrome.

1.5 Generic Solutions to 1-Off Problems

General solutions are more difficult and more costly to develop. A closer look range of problems with similar characteristics:

- Reuse is difficult to attain unless development is under taken for a range agent systems to be built, when you really need a one-off system.

Designing an architecture or testbed that supersedes multiple a

The ‘yet another agent testbed’ syndrome.

- Designing an architecture or testbed that supersedes multiple
1.6 Confuse Prototypes with Systems

Prototypes are easy (particularly with nice GUI builders!). Field-tested productions systems are hard.

Process of scaling up from single-machine multi-threaded Java app to multi-user system much harder than it appears.

Process of scaling up from single-machine multi-threaded Java app to multi-user system much harder than it appears.

1.7 Believe Agents

Silver Bullet

Holy Grail of software engineering is a "silver bullet": a order of magnitude improvement in software development.

Technologies promoted as the silver bullet:

- COBOL
- Formal methods
- Expert Systems
- Graphical Programming
- Logical Programming

Agent technology is not a silver bullet.

Reasons to believe that agents are useful in practice:

- Argument from analogy
- Argument from abstraction
- Argument from analogy
- Argument from abstraction

Process of scaling up from single-machine multi-threaded Java app to multi-user system much harder than it appears.

Field-tested production systems are hard.

Prototypes are easy (particularly with nice GUI builders!).

1.8 Confuse Buzzwords & Concepts

The idea of an agent is extremely intuitive.

Agent technology is not another abstraction.

Useful developments in software engineering: abstractions.

Abstractions in software engineering:

- BDI model
- Theory of human practical reasoning
- Agent architectures
- Serious applications
- Formal methods

Label "BDI" now been applied to WWW pages/Perl scripts.
Our system is a BDI system... implication that this is like being a computer with 64MB memory: a quantifiable property, with measurable associated benefits.

1.9 Forget it's software

Developing any agent system is essentially experimentation.

Forget it's distributed

But almost any principled software development technique is better than none.

Distributed systems = one of the most complex classes of problems.

Distributed systems = one of the most complex classes of systems.

Distributed systems tend to be distributed.

Multi-agent systems tend to be distributed.

Multi-agent systems to design and implement.

Problems of distribution do not go away, just because a system is agent-based.

Typical multi-agent system will be more complex than a typical agent system.

Recognize distributed systems problems.

Make use of DS expertise.

Forget it's software.

Developing any agent system is essentially experimentation.

Forget it's distributed.

But if almost any principled software development technique is better than none.

Distributed systems = one of the most complex classes of systems.

Distributed systems tend to be distributed.

Multi-agent systems tend to be distributed.

Multi-agent systems to design and implement.

Problems of distribution do not go away, just because a system is agent-based.

Typical multi-agent system will be more complex than a typical agent system.

Recognize distributed systems problems.

Make use of DS expertise.

Forget it's software.

Developing any agent system is essentially experimentation.

Forget it's distributed.

But almost any principled software development technique is better than none.
1.10 Don't Exploit Related Technology

In any agents system, percentage of the system that is agent-specific is comparatively small. Therefore, it is important that conventional technologies and techniques are exploited wherever possible. Don't reinvent the wheel. (Yet another communication framework.)

Exploitation of related technology:

- Speeds up development;
- Avoids re-inventing wheel;
- Focusses effort on agent component.

Example: CORBA.

1.11 Don't Exploit Concurrency

Many ways of cutting up any problem.

Experiments: compare functional, organisational, physical.

Serial processing in distributed systems:

- Only ever a single thread of control: concurrency one of the important potential advantages of multi-agent solutions not explored.

Parallelism is problematic. Many ways of cutting up any problem.

1.12 Want Your Own Architecture

Agent architectures: designs for building agents. Many agent architectures have been proposed over the years. If you don't exploit concurrency, why have an agent solution?

Great temptation to imagine you need your own.

Driving forces behind this belief:

- "not designed here" mindset;
- Intellectual property.

Problems:

- Architecture development takes years;
- No clear payback.

Before purchasing, ask: do you need an architecture?

1.13 Think Your Architecture is Generic

If you develop an architecture, resist the temptation to believe it is generic. Different architectures suit different problems. Any architecture that is truly generic is by definition not an architecture. It leads one to apply an architecture to problems for which it is not designed.

Example: The rising bread model of Winston.

If you do develop an architecture, resist the temptation to believe it is generic.
1.14 Use Too Much AI

Temptation to focus on the agentspecific aspectsof the application.
Result: an agent framework too overburdened with experimental AI techniques to be usable.
Fuelled by "feature envy", where one reads about agents that have the ability to learn, plan, talk, sing, dance...
Resist the temptation to believe such features are essential in your agentsystem.
The lesson: build agents with a minimum of AI; as success is obtained with such systems, progressively evolvethem into richersystems.

What Etzioni calls "useful first" strategy.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/

1.15 Not Enough AI

Don't call your on-off switch an agent!
Be realistic: it is becoming common to find everyday distributed systems referred to as multi-agent systems.
Problems:
- lead to the term "agent" losing any meaning;
- raises expectations of software recipients;
- leads to cynicism on the part of software developers.

Another common example: referring to a WWW page that has a multi-agent system:
- Be realistic: it is becoming common to find everyday distributed systems referred to as multi-agent systems;
- don't call your on-off switch an agent;
- pure A-O system = everything is in agent

More than 10 agents = big system.
Choose the right grain size.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/

1.16 See agents everywhere

"Pure" A-O system = everything is in agent, all systems = multi-agent systems.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/

1.17 Too Many Agents

Agents don't have to be complex to generate complex behavior.
Largenumber of agents:
- emergent functionality;
- chaotic behavior;

Lessons:
- don't have to be complex to generate complex behavior.

More than 10 agents = big system.
Choose the right grain size.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/

1.18 Not Enough AI

Don't call your on-off switch an agent!
Be realistic: it is becoming common to find everyday distributed systems referred to as multi-agent systems.
Problems:
- lead to the term "agent" losing any meaning;
- raises expectations of software recipients;
- leads to cynicism on the part of software developers.

Another common example: referring to a WWW page that has a multi-agent system:
- Be realistic: it is becoming common to find everyday distributed systems referred to as multi-agent systems;
- don't call your on-off switch an agent;
- pure A-O system = everything is in agent

More than 10 agents = big system.
Choose the right grain size.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/

1.19 See agents everywhere

"Pure" A-O system = everything is in agent, all systems = multi-agent systems.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/

1.20 Too Many Agents

Agents don't have to be complex to generate complex behavior.
Largenumber of agents:
- emergent functionality;
- chaotic behavior;

Lessons:
- don't have to be complex to generate complex behavior.

More than 10 agents = big system.
Choose the right grain size.
1.18 To few agents

Somedesignersimagineaseparateagentforeverypossible task. Others don't recognize value of a multi-agent approach at all. One "all powerful" agent. Result is like OO program with 1 class. Fail software engineers test of coherence.

1.19 Implementing infrastructure

There are no widely-used software platforms for developing multi-agent systems. Fail software engineers test of coherence. No effort devoted to agent-specifics. By the time this is developed, project resources gone! The result: everyone builds their own. To create a multi-agent system, requires basic infrastructure required agent systems. Such platforms would provide all the basic infrastructure required agent systems.

1.20 System is anarchic

Cannotsimplybundleagroupofagents.

Mostagent systems require system-level engineering. Cannot simply bundle a group of agents together. Organisation structure (even in the form of formal communication protocols) is essential.
1.22 The tabula rasa

- When building systems using new technology, often an assumption that it is necessary to start from a “blank slate”.
- Often, most important components of a software system will be legacy: functionally essential, but technologically obsolete software components, which cannot readily be rebuilt.
- Such systems often mission critical.
- When proposing a new software solution, essential to work with such components
- They can be incorporated into an agent system by wrapping them with an agent layer.

1.23 Ignore de facto standards

- There are no established agent standards.
- Developers often believe they have no choice but to design and build all agent-specific components from scratch.
- But here are some de facto standards.
- Examples:
  - CORBA;
  - HTML;
  - KQML;
  - FIPA.

2 Mobile Agents

Remote procedure calls (a) versus mobile agents (b):

- Why mobile agents?
  - low-bandwidth networks (hand-held PDAs, such as Newton);
  - efficient use of network resources.
- There are many issues that need to be addressed when building software tools that can support mobile agents...
  - security for hosts and agents;
  - heterogeneity of hosts;
  - dynamic linking.
Security for Hosts

We don't want to execute foreign programs on our machine, as this would present enormous security risks:

- If the agent programming language supports pointers, then there is the danger of agents corrupting the address space of the host.
- UNIX-like access rights on host.
- The agent might be modified (subverted!) in some way without our intent.
- We often do not want to send our programs, so to do so:
  - Trusted agents?
  - Some agent languages allow security properties of an agent to be verified on receipt.

Hosts must handle accessed programs cleanly — what do you tell an owner when their agent crashes?

Some agent languages allow security properties of an agent to be verified on receipt:

- Interpreted language
- Dynamic linking
- Use virtual machine technology

This implies:

- Interpreted language
- Dynamic linking
- Use virtual machine technology

Is clarity system dependent — reduced efficiency (performs complex language translations to machine code, which is clearer)

Secure co-processors are a solution — have a physically separate processor on which the agent is run, such that the processor is in quarantine (padded cell).

Some agent languages allow security properties of an agent to be verified on receipt:

- Interpreted language
- Dynamic linking
- Use virtual machine technology

Heterogeneity of Hosts

Unless we are happy for our agents to be executed on just one type of machine (Mac, PC, SPARC...), then we must provide facilities for executing the same agent on many different types of machine.

This implies:

- Interpreted language
- Dynamic linking
- Use virtual machine technology

Compiled languages imply reduction to machine code, which is clearly system dependent — reduced efficiency; (perhaps use virtual machine technology);

Dynamic linking:

- Use virtual machine technology
- Interpreted language

Trusted agents?

An agent might be modified (subverted!) in some way without our intent. We often do not want to send our programs, so to do so:

- Trusted agents?
- Some agent languages allow security properties of an agent to be verified on receipt.

Security for Agents

Agents have a right to privacy!

We often don't want to send out our programs, as to do so:

- Interpreted language
- Dynamic linking
- Use virtual machine technology

An agent can be protected in transit by using conventional encryption techniques (e.g., PGP). An agent can be protected in transit by using conventional encryption techniques (e.g., PGP).

Where is your knowledge of approval?

The agent might be modified (subverted!) in some way without our intent. We often do not want to send our programs, so to do so:

- Trusted agents?
- Some agent languages allow security properties of an agent to be verified on receipt.

Security for Hosts

Interpreted language to different environments. Interpreted language to different environments.
We can divide mobile agents into at least three types:

- **Autonomous Mobile Agents**: By autonomous mobile, we mean agents that are able to decide
  where (subject to certain resource constraints), when, and what to do when they get for themselves.

- **On-Demand Mobility**: The idea here is that a host is only required to execute an agent
  when it explicitly demands the agent.

- **Active-Mail Agents**: The idea here is to 'piggy-back' agent programs onto mail. The best-known example of this work is the mime extension to email, allowing Tinytalk scripts to be sent.

Autonomous

On-Demand Mobility

Active-Mail Agents
TELESCRIPT was a language-based environment for constructing mobile agent systems. TELESCRIPT technology is the name given by General Magic to a family of concepts and techniques they have developed to support mobile agent systems.

There are two key concepts in TELESCRIPT technology:
- places: widespread virtual locations occupied by agents. A place may correspond to a single machine, or a family of machines.
- agents: mobile programs, rather like TACL.

Agents are interpreted programs, rather like TACL. TELESCRIPT was developed to support applications that TELESCRIPT was not designed to support.

An engine is a kind of agent operating system — agents and places are executed by an engine.

- size (measured in bytes):
- lifetime (measured in seconds):
- money:
- money, measured in 'teleclicks' (which correspond to real

The most important resources are:
- what resources the agent can use;
- what the agent can do (e.g., limitations on travel);

TELESCRIPT agents have an associated permit, which specifies:
- completion time;
- destination;

which specifies the parameters of its journey.

In order to travel across the network, an agent uses a ticket.

Recall that:
- agents are mobile — they are able to move from one place to another, in which case their program and state are encoded and transmitted across a network to another place, where execution resumes.
- TELESCRIPT was an electronic marketplace application that TELESCRIPT was not designed to support.

Agents provide and consume a service of goods in the electronic marketplace: another: they occupy different places, then they can communicate across a network:

Agents can communicate with one another:
- they occupy the same location, then they can need one another.
Lecture 10

An Introduction to Multiagent Systems

Just as operating systems can limit the access provided to a process (e.g., in UNIX, via access rights), so an engine limits the way an agent can access its environment.

Engines continually monitor agents' resource consumption, and if an agent exceeds the limit, the engine kills the agent.

Engines provide (C++) links to other applications via Application Program Interfaces (APIs).

Agents and places are programmed using the TELESCRIPT language:
- Pure object-oriented language — everything is an object —
- Two levels — high (the visible language), and low (a semi-compiled language for efficient execution)
- Interpreted

Agents are ultimately interpreted programs, with a fairly powerful notion of agency;
- a rich set of primitives for building distributed applications.

Summary:

- A rich set of primitives for building distributed applications, with a fairly powerful notion of agency;
- Agents are ultimately interpreted programs, with a fairly powerful notion of agency;
- A rich set of primitives for building distributed applications.

General Magic claim that the sophisticated services make TELESCRIPT ideal for agent applications!

Summary:

- A rich set of primitives for building distributed applications, with a fairly powerful notion of agency;
- Agents are ultimately interpreted programs, with a fairly powerful notion of agency;
- A rich set of primitives for building distributed applications.

– a process class or which 'agent' and 'place' are sub-classes;
- semi-compiled language for efficient execution;
- two levels — high (the visible language), and low (a semi-compiled language for efficient execution);
- Interpreted;

Appointed based on SMALLTALK.

Languages:

- Ages and places are programmed using the TELESCRIPT application program interfaces (APIs);
- Engines provide (C++) links to other applications, so an engine limits the way an agent can access its environment.

Just as operating systems can limit the access provided to a process (e.g., in UNIX, via access rights), so an engine limits the way an agent can access its environment.
2.2 TCL/TK and Scripting Languages

The (free) ToolControl Language (TCL—pronounced tickle) and its companion TK, are now often mentioned in connection with agent-based systems. TCL was primarily introduced as a standard command language with graphical user interfaces. TCL programs are called scripts.

TCL programs can be executed by a shell program (tclsh or wish):

- They are plain text programs, that contain control structures (iteration, sequence, selection) and data structures (e.g., variables, lists, and arrays) just like a normal programming language.
- They can be executed by a shell program (tclsh or wish):
- They can call up various other programs and obtain results from these programs (e.g., procedure calls).

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them something to be desired:

- TCL programs are called scripts.
- TCL programs are interpreted, they are very much easier to prototype and debug than compiled languages like C/C++. TCL/TK also provides powerful facilities for interprocess communication, via the exchange of TCL scripts.
- TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

  - a free tool.
  - an interpreted language.
  - extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
    implemented in C/C++ and allows the user to build on these
    implemented features.
  - extended—TCL/TK is an interpreted language.
  - command language.

TCL/TK is an X Window System window manager that provides a set of primitives as a core of a set of primitives. TCL/TK can be embedded in C/C++ and allows the user to build on these implemented features. TCL/TK is an interpreted language. It provides facilities for command language.

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

- a free tool.
- an interpreted language.
- extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
  implemented in C/C++ and allows the user to build on these
  implemented features.
- extended—TCL/TK is an interpreted language.
- command language.

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

- a free tool.
- an interpreted language.
- extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
  implemented in C/C++ and allows the user to build on these
  implemented features.
- extended—TCL/TK is an interpreted language.
- command language.

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

- a free tool.
- an interpreted language.
- extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
  implemented in C/C++ and allows the user to build on these
  implemented features.
- extended—TCL/TK is an interpreted language.
- command language.

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

- a free tool.
- an interpreted language.
- extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
  implemented in C/C++ and allows the user to build on these
  implemented features.
- extended—TCL/TK is an interpreted language.
- command language.

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

- a free tool.
- an interpreted language.
- extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
  implemented in C/C++ and allows the user to build on these
  implemented features.
- extended—TCL/TK is an interpreted language.
- command language.

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

- a free tool.
- an interpreted language.
- extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
  implemented in C/C++ and allows the user to build on these
  implemented features.
- extended—TCL/TK is an interpreted language.
- command language.

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

- a free tool.
- an interpreted language.
- extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
  implemented in C/C++ and allows the user to build on these
  implemented features.
- extended—TCL/TK is an interpreted language.
- command language.

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

- a free tool.
- an interpreted language.
- extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
  implemented in C/C++ and allows the user to build on these
  implemented features.
- extended—TCL/TK is an interpreted language.
- command language.

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

- a free tool.
- an interpreted language.
- extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
  implemented in C/C++ and allows the user to build on these
  implemented features.
- extended—TCL/TK is an interpreted language.
- command language.

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

- a free tool.
- an interpreted language.
- extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
  implemented in C/C++ and allows the user to build on these
  implemented features.
- extended—TCL/TK is an interpreted language.
- command language.

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

- a free tool.
- an interpreted language.
- extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
  implemented in C/C++ and allows the user to build on these
  implemented features.
- extended—TCL/TK is an interpreted language.
- command language.

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

- a free tool.
- an interpreted language.
- extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
  implemented in C/C++ and allows the user to build on these
  implemented features.
- extended—TCL/TK is an interpreted language.
- command language.

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

- a free tool.
- an interpreted language.
- extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
  implemented in C/C++ and allows the user to build on these
  implemented features.
- extended—TCL/TK is an interpreted language.
- command language.

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

- a free tool.
- an interpreted language.
- extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
  implemented in C/C++ and allows the user to build on these
  implemented features.
- extended—TCL/TK is an interpreted language.
- command language.

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

- a free tool.
- an interpreted language.
- extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
  implemented in C/C++ and allows the user to build on these
  implemented features.
- extended—TCL/TK is an interpreted language.
- command language.

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

- a free tool.
- an interpreted language.
- extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
  implemented in C/C++ and allows the user to build on these
  implemented features.
- extended—TCL/TK is an interpreted language.
- command language.

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

- a free tool.
- an interpreted language.
- extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
  implemented in C/C++ and allows the user to build on these
  implemented features.
- extended—TCL/TK is an interpreted language.
- command language.

TCL/TK combined, make an attractive and simple-to-use GUI development tool. However, they have features that make them:

- a free tool.
- an interpreted language.
- extendable—TCL/Tk can be embedded in C/C++ as C++ code, which can be
  implemented in C/C++ and allows the user to build on these
  implemented features.
- extended—TCL/TK is an interpreted language.
- command language.
Sowheredoestheideaofanagentcomein? Itiseasytobuildapplicationswheretclscriptsareexchanged acrossanetwork,andexecutedonremotemachines. ThusTCLscriptsbecomesortofagents.

Akeyissueissafety. Youdon'twanttoprovidesomeonelse's scriptwiththefullaccesstoyourcomputerthatanordinary scriptinglanguage(e.g., csh)provides.

ThisledtoSafeTCL,whichprovidemechanismforlimitingthe accessprovidedtoascript.

Example: SafeTCLcontroltheaccessthatascripthavestotheUI.

ThusTCLscriptsbecomeagent-like. ThisleadtoSafeTCL,whichprovidesmechanismforlimitingthe accessprovidedtoascript.

Scopinglanguage(e.g.,csh)provides.

Akeyissueissafety.Youdon'twanttoprovidesomeoneelse's accessanynetwork,andexeutedonremotemachine.

Where.doestheideaofanagentcomein?