Language and semantics (1)	**Language and semantics (2)**
1. executable, 2. strong but flexible type checking, 3. measure function to guard recursions	1. syntax is relatively old fashioned (especially, SL), 2. union types without data constructors, 3. no parametric types
Abstract, Precise, Simple	Verbosity, Lack of Modern constructs (map, reduce,...), Lack of type inference
Simulation and abstraction capabilities, well suited for new DSL design	"Possible"-semantics can cause traps to users, language inconsistencies/mistakes are a real put off. The strong coupling between ++ and RT is a killer given that one might not want ++ and want RT features. Finally, there is no explicit support for refinement/reification, which is a crucial aspect of VDM! It would also be nice to have polymorphic parameters in types and/or modules (akin to what is possible say in PVS)
1) VDM-SL has a formal semantics 2) Changes can be requested via the Language Board 3) Easy to get started with (compared to other formal methods)	1) VDM++/VDM-RT semantics are poorly defined, which complicates verification 2) VDM++ and VDM-RT are feature-wise too complicated for formal languages (inheritance, overloading, union types)
Relatively easy to learn - the are no unusual concepts. Easy to read simple models (in ASCII), even for non-experts. It does have a track record (not new and unknown)	Specification of threading and read-time always feels awkward to me. The detail of OO semantics is still poorly defined. It is probably perceived as an old language/method = irrelevant?
1) The language board makes sensible analysis of potential adjustments of the VDM notations 2) The semantics of the VDM languages is explained and exemplified in the language manual 3) Test cases are present in the Overture tool analysing different semantic constructs	1) The formal semantics of the VDM languages has only really been described for (an older version of) VDM-SL 2) The semantics of the VDM languages is not present in a form that can be used as a basis for theorem proving support 3) Test cases are not as exhaustive as they could be for different semantics aspects
1) Unambiguous definitions of types 2) Easier to express operations that are otherwise difficult to express in anything other than natural language	1) There is no way to easily handle structured data that is represented in the real world by bit fields. 2) One-based indexing of sets and sequences is annoying when the rest of the world (mostly) uses zero-based indexes.
* Data representation and abstraction * Readability * We care about semantics	* Three dialects is confusing; lack of e.g. real-time SL * No proof in ++ / RT * RT CPU/BUS/scheduling doesn't always match real-world usage
1. A low entry level for the basic language. 2. A clearly defined executable subject. 3. Breadth of applicability to systems as well as software.	1. It is so comprehensive it appears complex: the o-o, concurrency and RT extensions make it seem baroque. 2. The relationship to commonly used languages either for programming or model-based engineering is not well defined as "part of the package"
Programmer-friendly syntax, Functional flavour, Undefinedness	Object-Oriented Semantics are very poor, language is too big, type system is too loose in some ways and too restrictive in others (ex: high-order function polymorphism)
(1) Programmer-friendly syntax including object-orientation, e.g., Map rather than Relation (rather than all very mathematical notation), (2) simple but therefore clear control of abstraction by the two levels of implicit/explicit specification, and (3) still evolving language	(1) obsolete if we admire the "programmer-friendly syntax and OO", such as lack of generics in Java (polymorphic functions, but not for operations) and no "atomic" for state change by calling operations ("atomic" only for assignment statements), (2) unclear or imperfect consideration on semantics regarding consistency (e.g., the recent addition of "pure" operations, definition, the atomic issue mentioned above), (3) and instability due to #2 point (syntax change and tool-warning/error change) .
(1) extensive language reference guide and many working examples (2) covers all dialects supported by the tools (3) no significant deviations between Overture and the ISO standard	(1) language is very big with many constructs that are considered obscure and typically are poorly understood by novices; the consequences of a type system with looseness is hard to grasp and needs to be addressed with care, in particular in applications that require high levels of rigour (2) the object oriented aspects are weakly defined and not in line with other OO languages, causing difficulties and misunderstandigs; we should consider if OO is actually needed at all (3) the appeal factor of modern functional languages is missing from VDM which seems a turn-off for some new user; it would be good to benchmark the language against some alternate approaches to see if we need to prune and modernise? do we really need all these language variants? (4) the fact that we lost the ability to bootstrap our tools from a formal specification of the tools itself I think is still one of our main weaknesses - a capability we basically lost about a decade ago. IMHO it would add to our credibility and challenge our tool development if this was reinstated!

Tool support (1)	**Tool support (2)**
1. Eclipse is a common platform that can be chained with the implementation phase, 2. syntax checking as you type, 3. cross platform among different OS's.	1. no pretty printer (auto format), 2. occupies a large area on screen, 3. resource demanding
Standalone IDE, Good Debug Env, Test features	Frustrating bugs, Low performance, Lack of advanced IDE features (eg: refactoring,...)
The community is fantastic: open minded, responsive, collaborative. Problems raised are answered/dealt with as best as possible.	Overture doesn't seem to scale well for larger models; profiling capabilities within the tools would be nice. The POG is flaky/weakly defined.
1) Easy to use 2) Supported by a lively community 3) Good documentation (user manual, language reference manual)	1) Lack of verification support 2) Auto-completion/refactoring support should be enhanced 3) Prototype features are rarely finished/matured for releases
The Eclipse UI is relatively familiar and easy to use. Bugs and problems are generally looked at quickly (and hopefully fixed!) It is stable: it is rare for Overture to crash.	There are few other users, so it may feel risky to adopt. There may be scalability problems (eg. with the large EMV model) No tool help for refinement or formally linking a model to an implementation.
1) The division between a core part and a GUI part is important for its future development 2) The animation/debugging feature and its ability to incorporate legacy code is absolutely essential 3) The ability to produce FMUs from Overture so it can be used in a co-simulation context	1) The lack of theorem proving support for in particular VDM-SL (the others are not feasible in a reasonable amount of time) 2) The Eclipse GUI is getting old and here IntelliJ GUI would be a better future alternative 3) It would be great to have additional support in a cloud context
1) Navigating around large models is fairly easy to do. 2) Being able to execute models and debug them as you would a program is great 3) Picking up errors is easy	1) Calling external DLLs isn't straightforward, but is essential in large projects.
* Robust, doesn't crash too often * Easy to install for single users * Plenty of examples	* Lack of auto-completion * Error messages can be esoteric * Hard to deploy on network clusters
1. It's comparatively well documented 2. It's comparatively robust 3. The community behind it is quite welcoming	1. Its continuity depends on a small number of people who will not be there forever 2. It doesn't take account of advances in formal techniques such as model checking and proof support 3. It is not regarded as a vehicle for research in formal methods (if you care about that!)
Extensibility of tools, good debugging support, code generation	no theorem proving, very few reusable libraries, eclipse platform is a bit old
(1) active updates, (2) good support for interpreter-related functions, and (3) good modernization via integration with the Eclipse platform	(1) naive implementation and frequent updates on consistency checking (such as issues written in the "Language and semantics (2)"), (2) weak implementations of non-core but notable functions, such as UML generation (no work with the latest Modelica) and proof obligation generation (in a recent version it said "--after execution\n$postcond", now it says $precond => $postcond without using the weakest precond, which is a wrong formula), (3) weak support of concurrency (no model checking) though VDM++ has concurrency
(1) we have two strong and high quality toolchains available to the community free of charge (2) tools are available on Windows, Mac and Linux (3) tools are well maintained and regularly updated and performance is overall OK	(1) the tight coupling with Eclipse is both a blessing as well as risk; Eclipse is here to stay but it seems to loose momentum and interest (2) the GPL license model is considered restrictive to enable commercial initiatives; perhaps BSD two-clause should be considered for the next generation of tools? (3) we do not exploit the power of cloud computing (4) the implementation of VDM-RT with fixed time step is inherently flawed and should be addressed (5) the performance of the interpreter and quality of the code generators is not good enough to deal with the next generation of problems

Applications (1)	**Applications (2)**
1. virtual machine, 2. embedded (felica), 3. education	1. virtual machines, 2. IoT, 3. Automotive systems (car management, navigation and entertainment)
Precision agriculture, Education	Education
Security protocols (AnB language semantics), Payment systems (EMV, PSD), Computational games (TicTacToe, DotsAndBoxes, Suduku, Ultimate TicTacToe), Flash memory management, Garbage collection algorithms	Not sure. Perhaps examples to explore POG and Refinement chains? Or integration with provers/model checking tools?
Harvest planning (the AGCO project)	Online courses
EMVCo model (card payment systems). The MULTOS model (smartcard OS, work suspended now?) Several models of UK Post Office systems, used internally in Fujitsu. Models of air traffic reporting protocols (Airservices Australia) AGCO harvesting models Some models inside ESA?	Other standards bodies (after Airservices example)?
1) It has been used by Nick for commercial applications with critical aspects of administrative systems at Fujitsu (in the past) 2) It is used for agricultural logistics the company AGCO is using it 3) It is used for teaching purposes at a number of universities	1) It would be good to see it used on the Mobile Felica cases 2) In general it would be good to use it in the development of critical systems 3) Increased use in a co-simulation setting in the development of CPSs
Attempting to model a secure operating system	Modeling of secure applications
.	.
1. Controller design (through multi-modelling and cosimulation) 2. Education & Training	1. Comprehensive model-based engineering and analysis of digital aspects of larger systems (infrastructure, buildings, etc.) 2. Providing semantics for domain-specific languages in areas like the above, as well as IoT areas.
critical systems, embedded control, distributed systems	AI, critical systems even more
Rigorous specification (rather than strong checking, like in Felica)	Potentially, as a simulation language as in Crescendo and INTO-CPS, but with more powerful tool support as a "simulation" tool rather than a "specification" tool
reactive systems, embedded control systems	autonomous and self-* systems

Community building (1)	**Community building (2)**
slowly growing	Create open-source projects that use VDM and put VDM specs with C/C++/Java/Swift/what-so-ever program source on github. Not spec only, but spec with program source.
Good	Promote efficient tools
The community is active and friendly.	Some processes for language discussion, tool improvement, etc can be quite a drag/overkill.
Doing fairly well: workshop on an anual basis (more or less) and we do see new people joining the community now and then	We need to advertise the language/tools more. For example through online courses or by encouraging people to use join discussions on Github/StackOverflow/other kinds of online groups. I would prefer this over the existing mailing lists.
It is still largely academic, with just a couple of industry contacts. The support email lists and StackExchange tags are rarely used. Downloads are good, but we don't know who they are (may be bots).	Explicitly offer support for new users? Support the EMV project (Leo/Martin) as much as we can.
1) primarily mouth to mouth for key stakeholders 2) whenever new releases are produced emails are sent around to a mailing list of users 3) passive via web pages and manuals	1) coursera courses of how to use VDM and Overture 2) Proactive marketing towards potential users 3) improved on-line videos about overture at youtube disseminated via facebook and twitter
Great tools and output. However I only found out about Overture by accident...	Is there an Overture Tools users forum somewhere? It would be good to build a knowledge base out there of how to do things...
Growth is slow but it's happening.	* Ask new people (Paul, Simon etc.) how they found us; do more of whatever they suggest. * Submit to things like F-IDE or other communities
We are open and welcoming; the workshops are inclusive and stable. We are not necessarily maximising the assets we have by using it to support research in systems engineering in a variety of new areas (just look at the Gartner hype curve!)	The community relies on voluntary work, mainly by academics who are necessarily motivated by research on the fundamentals of computer science and systems engineering. However, we have a strong focus on achieving industry use (a great goal that we must retain). As a result, our focus has been on delivering a platform that works in industry case study settings (in EC projects) and perhaps less on offering a platform for research (integrating and adding analytic capabilities, extending expressiveness of models, etc.). I'd suggest that we have a small research working group report back to the community on how Overture might be developed as a platform for research activity, so that there is enough of a "pay back" for community membership.
The community is good, but has not grown in a long time, which can make things a little stale	Modernize the tool platform and encourage open source contributions, show that overture can solve problems other tools cannot
Very strong and active, I like very much	Include the state-of-the-art from a much wider-viewpoint: looking at other tools and other paradigms (trends and techniques in general software engineering), rather than a bottom-up improvement
very active core team with strong ties in academia, however, we have never been able to grow very much - but this seems common to all FM tool communities	IMHO the only option is to join forces with other FM groups and perhaps be more active on social media (we are quite passive in our communications)

Business offerings and long-term support (1)	Business offerings and long-term support (2)
Create open source that use VDM. Such projects would work as reference for managers who are concerned with quality of specification.	Eclipse will not live forever. We'll anyway need to envision new VDM that adopt new styles of software development and create a new tool that supports new VDM and future programming languages on top of future development substrates.
Provide industrial partners with added value to their dev. processes	Find key partners
It must scale, and scale well. Support is good. Examples of different uses would be good. Also examples of "ways not to do it" that are frequently caveats for inexperienced users?	Mixture between industrial clients and academic use.
Getting more companies involved through research projects	Getting more people interested in the language/tools via courses and research projects.
Advertise significant industrial success (EMV) - ideally from EMVCo themselves. Give interviews to industry press about industrial uses?	Short term academic projects and voluntary support (as now). Medium to long term, consultancy fees paying for tool experts.
1) list potential consultancy suppliers at the overture web pages 2) offer commercial support of the Overture tool for potential users 3) present examples where Overture/VDM has been used in the past	1) additional externally funded research projects where Overture is included as a part 2) increase the value in contributing to the Overture open source development 3) get at least one company offering commercial services on top of the Overture platform
Explain the financial benefits of modeling and provide more examples (well documented ones at that which explain each step in detail).	The tools need to be free (at least for now) as having to pay would only be another reason for business not to take that first look. A donation system could be tried...
* Research and innovation projects (EU, Innovate UK) * Champions in key industries * Success stories, articles etc. in industry magazines (not just scientific venues)	* Research and innovation projects (EU, Innovate UK) * INTO-CPS Association
Integration with established tools and methods - even using VDM/Overture as a semantic foundation rather than as a first class tool in its own right? Develop links with tools developers rather than "compete" by trying to make a robust industry offering ourselves. Otherwise we will only have Overture as the tools that's used as a basis for consultancy.	Set ourselves a goal (5-year) to establish a sustainable association in the INTO model? (Sorry running out of time here)
Successful case studies, commercial support, ability to solve problems	engagement with open source enthusiasts
Modern and active tool support, success story by a strong industry (like one by Felica, which made VDM the most popular formal specification language in Japan)	Active community and relevant state-of-the-art research funding
To provide a solution to industrially sized problems, so reliability, scalability and performance of the tools are key - closely followed by accessibility and low upfront investment cost. The threshold to start should be near zero (even tool installation can be considered a barrier), with low-hanging fruit that addresses some urgent and recurring business need; these might well be secundary to us (e.g. automatic documentation generation) but are key to acceptance in industry. It would be good to spend some time to investigate with are the key success factors from industrial perspective? Do we know and do we have an answer to that?	Be responsive to queries and help industrial users to overcome these hurdles. Perhaps some level of additional support can be offered as a commercial service, i.e. to implement new features, solve modelling issues, etc. Some modelling market place?