AN INSIGHT WHITE PAPER

From Assistant to Autonomy: Navigating the Three Levels of AI in Software Engineering

Executive summary

If you are responsible for custom built business systems, your world is turning upside down. AI is revolutionizing the business of software engineering. The practices you've been using for 25 years must now be completely re-designed.

Software Engineering Services

Some custom software is built in-house, but $200 billion per year is spent contracting out software engineering to consultancies and outsourcers. Very few contracts are based on results, though. Most are priced based on hours consumed.

AI and Software Engineering

Since 2020 there has been an explosion of tools which use generative AI for software engineering. OpenAI, Google and Microsoft have all produced tools which can generate code; and there is a host of fast-growing startups with competing offerings.

Business Impact

AI can save 40%-50% of software engineering effort. (Though reports vary - in some situations the benefits are much smaller).

That means there is a huge opportunity to deliver business change faster. Requests from the business should no longer have to wait in a backlog for months. Teams can deliver working software with new substantial features in days.

To get there, however, we must change the way the business works with software engineers. We must make software engineering teams smaller and re-think the roles in an agile team.

The Next Ten Years

We expect that AI adoption for software engineering will progress through three levels:

Level 0

- No generative AI
- Normal before 2025
- Agile teams of 4-9 people from different disciplines
- Agile methods used for good communication between people
- Tool cost covers dev test and deployment
- Scripted automation of testing and deployment

Level 1

- AI Assistants
- Common since 2025
- Similar composition of teams
- Same agile methods
- Each team member uses an AI assistant to accelerate their own part of the process
- Tools cost covers dev test, deployment and AI assistants

Level 2

- Autonomous A1
- Dominant by 2040
- Fewer humans in the team
- More seniors, providing high level instruction to AI. (fewer juniors)
- Agile methods must be re-engineered (fewer people)
- Tools cost covers dev test, deployment and AI assistants

Level 0 represents conventional agile methods, as set out in the Agile Manifesto back in 2001. There's lots of automation, but no generative AI.

Many organisations have already moved to Level 1. AI acts as an assistant to agile team members, but accountability still lies with humans. At Level 1 agile team roles and processes are unchanged.

Ambitious organisations are moving rapidly to Level 2. Most work is done by autonomous LLMs, supervised by a small number of senior people. That means software engineering teams will become smaller.

A new role must be created, which we have termed Product Creator. Agile coaches must redesign workflows, ceremonies and documentation. The benefits are clear. However many organisations are constrained by a mix of inertia, skills shortages, restrictive policies, and services contracts which were designed for Level 0.

Call To Action

Everyone in the software engineering world is affected by AI. This paper shows how to overcome the constraints and use AI to bring a competitive advantage to your software engineering teams.

Software Engineering Services Today

The Market

The market for software engineering services is growing at 8.9% a year and is expected to exceed $238 billion in 2028. (Gartner, 2024)

Clients buy custom software development because they have a unique need which cannot be met by commercial-off-the-shelf enterprise software suites. Examples are:

- Digital products and platforms;
- Unique back-office processes;
- Software embedded in products such as cars and TVs.

There are hundreds of consultancies and outsourcers in this market, ranging from 20 person specialist development shops to giants like Accenture and TCS.

Services are often delivered remotely from offshore locations such as India, or from near-shore locations such as eastern Europe or Latin America.

Some clients buy individual software engineers and mix them in teams with their own developers. Others buy whole teams, or sets of whole teams.

Paying for Results

When clients contract out work, they normally like to pay based on value received.

However software engineering services has remained stubbornly resistant to this.

There are five pricing models in the industry:

a. **Fixed Price.** Most procurement professionals prefer fixed price, because the supplier carries the risk of getting work done on-time and to budget. However fixed price also means fixed scope. Most software teams use agile methods which encourage changing scope as often as the business needs. This makes true fixed price/fixed scope contracts all but impossible.

b. **Time & Materials (rate cards).** The supplier provides a table showing the price for each person on the team – usually per day, sometimes per hour. T&M is well-understood and very flexible. It is the most common way that contracts are priced – but the client is paying for effort, not value delivered.

c. **Managed Capacity (squads/pods).** This is a variation on time & materials, most often used when a supplier provides an entire team. A fixed fee is charged (per month or sometimes per sprint), based on an agreed team structure. Again though, the client is paying for effort, not results.

d. **Unit Pricing (stories/story points).** Agile teams often count stories or story points to size work. If clients paid per story point, this would be more like paying for results. However few contracts use this method. Buyers are put off because there is no industry standard definition of a story point.

e. **Outcome Based.** The supplier is paid a percentage of the business value that the client achieves from their work. Some suppliers will agree to this pricing model, particularly if they know the client well and feel confident that the business benefits planned are realistic. However, these schemes have never really taken off.

An Opportunity to Improve Pricing Models

The introduction of AI in software engineering is an opportunity to change industry pricing models. Clients should no longer pay 'per day', because the supplier is providing both days human effort and also AI compute capacity.

This creates an opportunity for clients and suppliers to focus on what is important – the business value delivered – and structure payment schedules around business impact instead of labour provided.

AI and Software Engineering

Software Development with Large Language Models (LLMs)

OpenAI released GPT-3 in 2020. They were surprised to discover that people started immediately started using it to write software. Five years later, AI-assisted software engineering had built up huge momentum:

- GPT-3 has expanded into ChatGPT, and the technology has been built into Microsoft and Github's Copilot tools.
- Google's has introduced Gemini to compete with ChatGPT, and there has been an explosion of new LLMs (Large Language Models) such as Anthropic's Claude, Anysphere's Cursor, and Amazon Q.
- A new developer culture is arising, "vibe coding", in which you "forget that the code even exists" … I "Accept All" always … I just see stuff, say stuff, run stuff, and copy paste stuff, and it mostly works" (Andrey Karpathy, Feb 2025)
- The founder of Instagram predicts that by 2028, software engineers will spend more time reviewing AI-written code than writing it. (Mike Krieger)
- Large software engineering organisations have announced an intention to invest in AI first, people second (Google, Microsoft, DuoLingo, Shopify)
- Developers who were spending $100/year on AI tools in 2023 had increased spending to $5,000/year by 2025 (Grant Slatton)

The pace of innovation over the next few years will be rapid. Each new release of an LLM brings new and often unexpected capabilities. Choice is a challenge. How do you know which tools will win and which will fail? Tracking the continued growth of these tools requires continual focussed research. The impact extends far beyond simple coding, to everyone in an agile software development team.

Product Owner

Gathering Requirements

The Product Owner represents the needs of many stakeholders, creating and prioritising the team's goals and maintaining the Product Backlog. (Scrum Guide). Product Owners can use AI to replace traditional notetaking with meeting transcripts and summarisation.

Tools examples: Teams, Zoom, Otter.AI, LucidChart, Miro AI

Proxy Product Owner

Writing Roadmaps, Epics, Features, and Stories

Strictly speaking, user stories should be written by the Product Owner. However, when Product Owners are short of time, they sometimes use an assistant to document the detail, known as a Proxy Product Owner. General purpose LLMs can generate text in user story format, and so can specialist tools for managing software development.

Tools examples: Jira AI, ProdOps.AI, ChatGPT and Google Gemini

Human-written stories can contain gaps or inconsistencies which come to light during development causing delays and re-work. We don't know yet whether AI-written stories will have more of these, or less. A scanning tool such as ScopeMaster could be used to assess the quality of AI-written stories.

Scrum Master

Sizing and Estimating

Faced with a set of stories to develop, the team must prioritise them and assign them to sprints. To do this the Scrum Master needs to estimate how long each story will take. Story points is the most widespread technique for doing this.

Recommended practice is for story points to be agreed by consensus between product owner, scrum master, and the team – ideally by comparing new stories against a set of reference stories.

Some people have tried using LLMs to award story points. For sure, LLMs can come up with a number. However that number may not be accurate – and it may not be trusted by the developers. Jeff Sutherland, one of the creators of Scrum, writes that using ChatGPT and Otter.AI for a first estimate reduced the time needed to estimate story points from 45 minutes to one minute. However he also writes: "AI is already helping reduce manual backlog work, but Scrum teams still need to guide prioritization and oversee refinement sessions".

Tools examples: ChatGPT, this academic study.

Change Impact Analysis

LLMs can write Project Initiation Documents (PIDs) which structure requirements, analyse existing database schemas, propose changes, and write code to migrate to the new schemas. (Source: Justin Campbell)

Designer

User Personas, Customer Journey Maps, Wireframes, and Mock-ups

Designers consider the people who will use the system, imagining their workflow and creating the screens and forms they will use.

LLMs can be used to speed up the creation of user personas, customer journey maps, wireframes, mock-ups, and prototypes. LLMs help designers to experiment –adjusting layouts by changing the LLM's prompt is quicker than redesigning an entire form.

Tools examples: ChatGPT, Figma AI, Miro Assist, Uizard, Visily, Vercel

Architects

Technical Architecture and Integrations

Although Technical Architect is not a role specified in the Scrum Guide, in practice many software engineering teams need to submit their technical architectures for governance approval.

LLMS can produce a substantial first draft of a technical architecture document, to be finalised by the architect. Furthermore, few applications are standalone. Most connect to other systems, drawing data from them or supplying data to them. LLMs can compare input and output database schemas, match fields with the same (or similar) names, and write transformation code to convert data types, clean data, and use external APIs.

Tools examples: ChatGPT, Google Gemini, Claude, Jiffy.AI, Altova, Blocstop.

Developers

Writing Code, Or Helping To Write Code

The core task of writing code is where LLMs are most advanced.

One of the earliest tools was Tabnine, a code completion tool written in 2018. Early versions would suggest how to complete a line of code – a bit like a smartphone suggesting the next word in a sentence. LLMs have now advanced to the point where they can write entire blocks of code – like Google Mail writing a whole paragraph.

Experience software engineers use these tools to speed up the parts of coding that they find boring, such as repetitive, lengthy blocks of code. Less experienced software engineers use LLMs as a training aid, showing them how to write code in a language they don't know well. In 2025, the term "vibe coding" became widely used to describe building software with an LLM without even reviewing the code it writes – just testing each iteration, and revising the prompt, and trusting the LLM.

Software engineers report mixed results from AI code-completion tools. Most have tried it. Some developers

are overwhelmingly positive; others report that first drafts were produced in an instant, but then the LLM failed utterly. Reported efficiency gains range from near zero to over 50%. A 2023 McKinsey study suggests the gains are greatest for senior developers and simple tasks, but zero or even negative for junior developers and complex tasks. One intriguing use of LLMs is in code reviews, or even as one of a pair in pair-programming.

The results may be mixed, but the costs are certainly growing. One blogger reports that they spent $100/yr on LLMs in 2024, but $5000/year in 2025. $5,000 per year is around 5% of the cost of a mid-senior onshore developer – but it is 20% of the cost of a junior offshore developer. Some tools are pay-per-use, so fast growing adoption can create unpredictable costs.

Tools examples:

· the LLM giants - OpenAI's ChatGPT and Google Gemini;

· the coding tools giants - GitHub Copilot (by far the most popular according to a 2024 survey by Pragmatic Engineer), Microsoft's related CoPilot, Oracle, AWS CodeWhisperer);

· and hundreds of start-ups – Tabnine, Anthropic's Claude, Cursor, Replit and Devin.ai to name a few.

QA/Test Engineers

Testing

The more that LLMs are used to create software, the more important it is to test. AI is known to sometimes "hallucinate", producing bizarre results. Thorough testing makes sure defects don't slip into live use where they could damage the business.

LLMs can automate some of the basic work of testing, such as generating test data or finding gaps and oversights in test scripts. A more sophisticated use of LLMs would be to generate Gherkin scripts, managing the test library (identifying redundant or out-of-date tests, analysing results from a large test run, or even helping to find the line of code that is causing a test script to fail.

Testing is often treated as the poor relation in a software engineering team. The work is less glamorous than coding. And while thorough testing is always appreciated, the delays caused when testers find a problem and code needs to be re-worked are less popular. Using LLMs to make testing more efficient will not produce more software. However it ought to produce software which works better.

Tools examples: DiffBlue, Functionize, Mabl, Tricentis, Parasoft, SmartBear

DevOps Engineers

Managing Deployments

DevOps engineers are responsible for creating automated pipelines for continuous integration and continuous deployment. This enables new code to be released live frequently and with little delay.

The output of a DevOps engineer is primarily text: scripts, package definitions, and code. Therefore all the tools used by software engineers can be used here as well.

Tools examples: Harness.IO, Ansible Lightspeed

Support & Maintenance

Finding and Fixing Problems

Once code has gone live, the support engineer's role is to take issues raised by users, work out whether there is a problem in the code, and fix the code if necessary.

LLMs can read code and summarise what it is doing, even writing comments explaining what the code is doing. Newer reasoning models can be given a block of code and asked to find a problem (for example at AWS and Microsoft).

Sometimes code has been edited and re-written so many times that it becomes hard to maintain. It needs to be re-written from scratch, or 're-factored'. LLMs are developing a good reputation for re-factoring, freeing up dev time.

When an issue is found in software, business users usually want it fixed as soon as possible. A tool which can speed up a large part of that process will mean problems are solved much faster. Using LLMs for support and maintenance could produce the most visible gains for buyers of software engineering services.

Tools examples: Claude Sonnet, Giskard AI, Phind, Llama.

Three Levels of AI-Assisted Software Engineering

Level 0 – No AI Before 2025

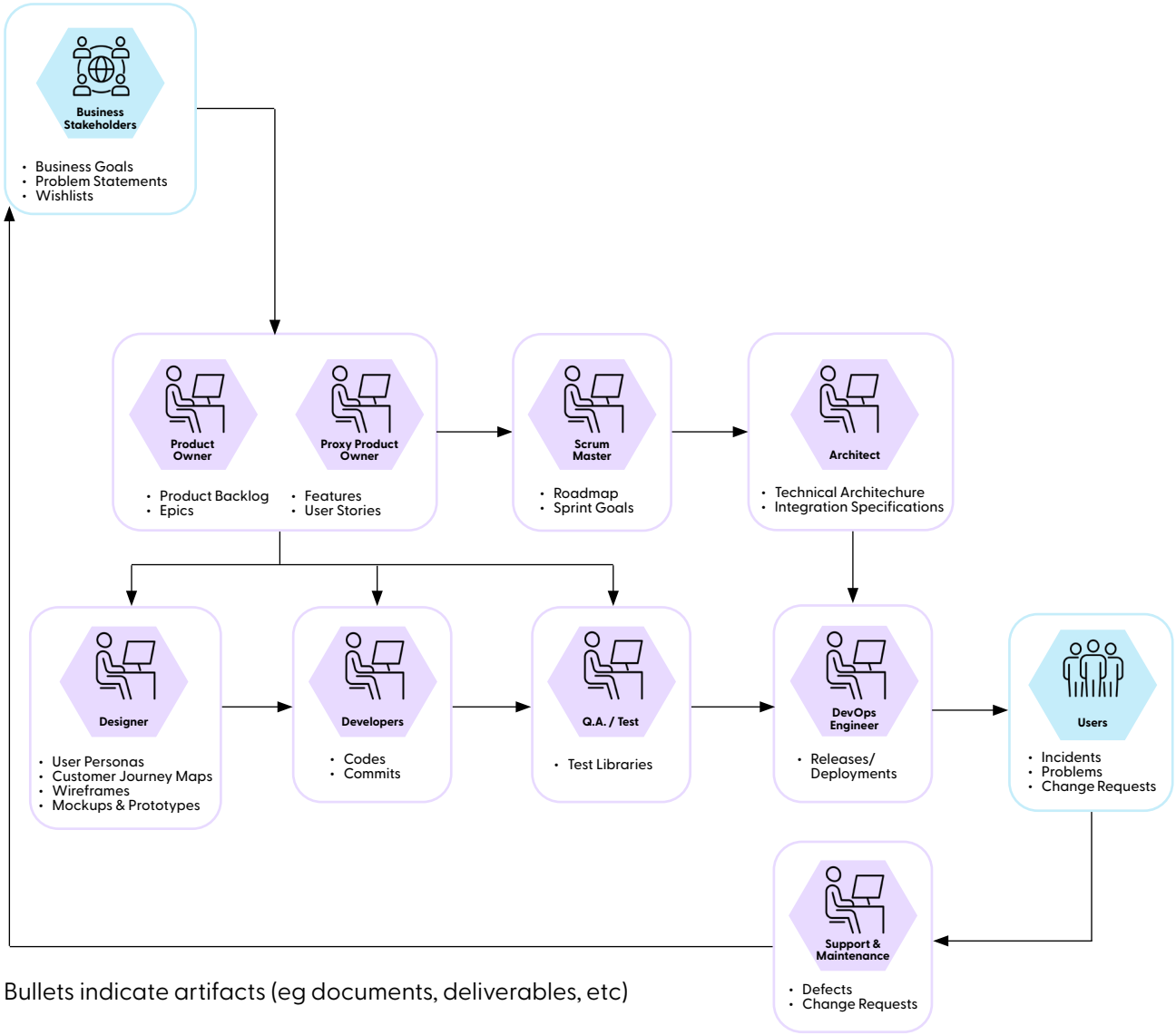
The way that software engineering has been done for most of its history involves plenty of automation - but not artificial intelligence.

The early parts of the software development lifecycle have been human work, done by skilled creatives. User stories are written by Product Owners. Customer journeys are drawn up by designers. Code is written by developers. Testing is done by QA Engineers.

Once code has been written, high levels of automation have become common. Manual testing has been replaced by automated test scripts, often run overnight as part of nightly builds. Manual releases have been replaced by continuous integration & continuous delivery (CI/CD).

This kind of automation requires a script to be written by a skilled engineer. There is no artificial intelligence. Everything is created by humans, even the automation.

Agile methodologies like the workflow shown is intended to ensure clear communication between humans.



Bullets indicate artifacts (eg documents, deliverables, etc)

Three Levels of AI-Assisted Software Engineering

Level 1 – AI Assistants 2025

2025 marks a turning point. The number of AI tools has exploded. Most software engineers have already experimented with them. Now product owners, designers, and testers are also trying out AI.

Many are sceptical, and with good reason. LLMs often produce a good first draft which turns out not to scale up. Demonstrations tend to be on simple applications – it's not clear how well AI works on very complex software. The quality of AI-written code can be poor. AI is known to sometimes hallucinate – people are concerned that this should not inject hidden bugs into live systems.

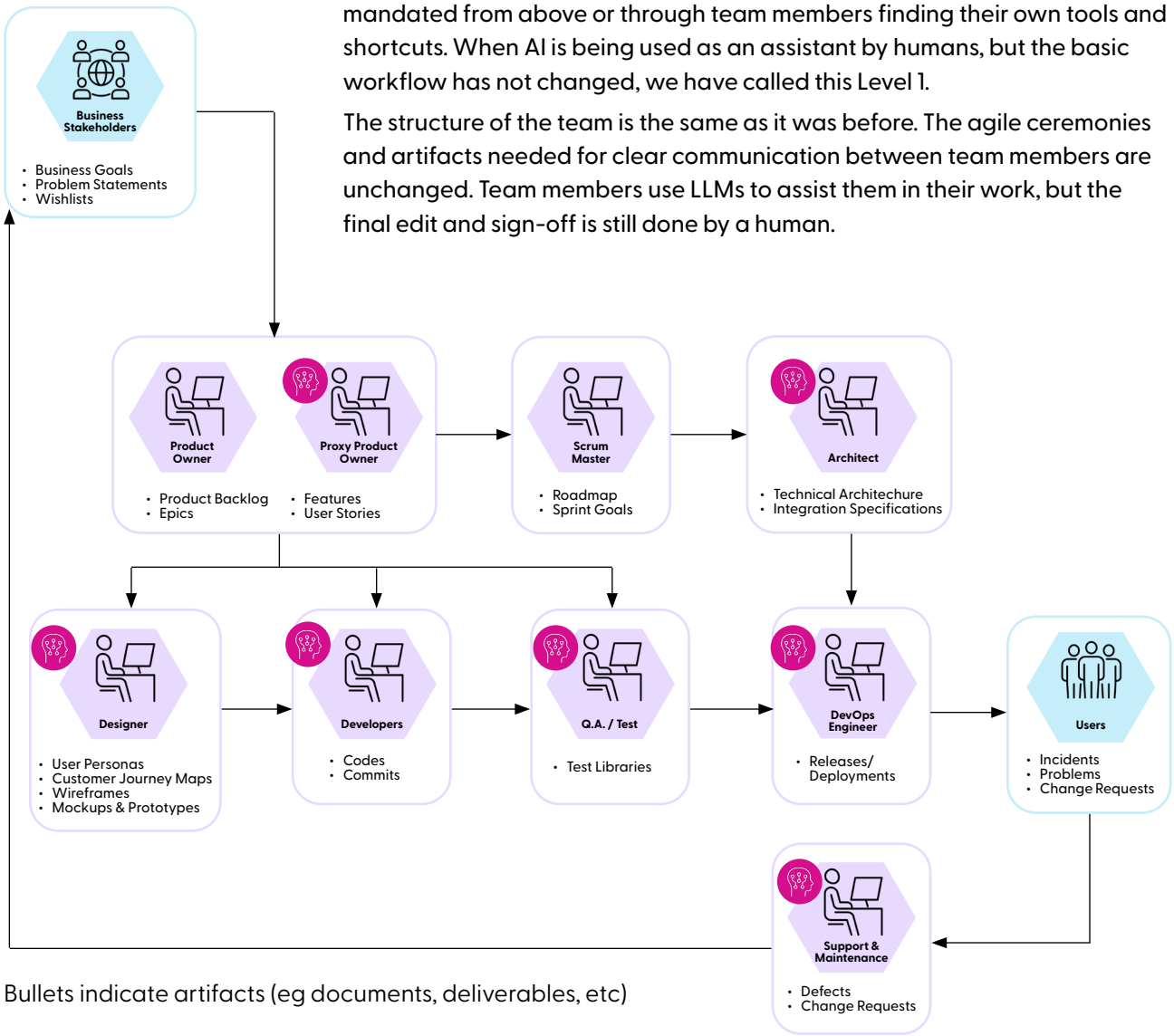
However, the forces driving adoption of AI Assistants are powerful:

- A constant thirst from the business for IT to do more, faster
- The prospect of substantial improvements in productivity and time-to-market
- Enthusiastic momentum from tech innovators and early adopters
- A weariness with the perennial challenge of finding, training, and retaining software engineers.

Level 1 – AI Assistants

The use of LLMs will certainly grow over the next ten years, whether mandated from above or through team members finding their own tools and shortcuts. When AI is being used as an assistant by humans, but the basic workflow has not changed, we have called this Level 1.

The structure of the team is the same as it was before. The agile ceremonies and artifacts needed for clear communication between team members are unchanged. Team members use LLMs to assist them in their work, but the final edit and sign-off is still done by a human.



Bullets indicate artifacts (eg documents, deliverables, etc)

Three Levels of AI-Assisted Software Engineering

Level 2 – Autonomous AI Beyond 2025

As teams work more with LLMs, they start to rely on them. This triggers a re-engineering of the workflow within an agile team which we have called Level 2.

Level 2 – Autonomous AI

A new role is emerging, which we have called Product Creator. This role combines the skills of a Product Owner and a CTO. It is similar to what some call a "Product Engineer", or the Forward Deployed Engineer role seen at OpenAI and Palentir – a senior software engineer who is embedded within a customer operation to learn at first hand how the business works.

The Product Creator provides the vision for how the system will benefit the business. They produce a Product Description which is issued to a network of agentic LLMs who deliver the software. The AI agents do the design and development work, and will be autonomous – supervised by humans rather than assisting them.

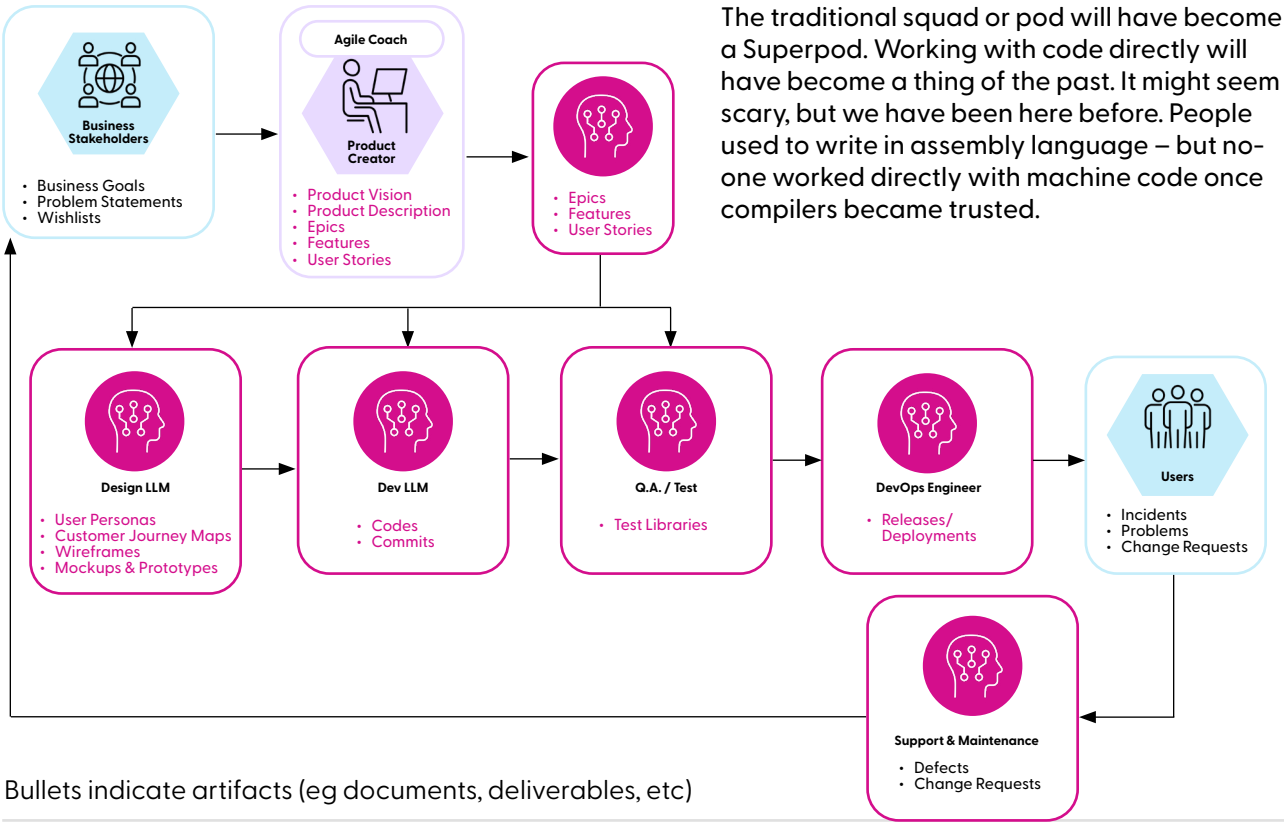
Parts of the workflow shown (pink bullets) can then be re-thought. There will be fewer people, but those people will be more senior and will combine deep technical knowledge with a close understanding of the business.

There will be no need for a conventional Scrum Master. The agents work so fast that a person is not needed to maintain the flow of tickets between people. Autonomous agents can even sequence and prioritise some of the work in the way a Scrum Master does.

The Architect role could be combined with Product Creator for small stand-alone projects. But in most large complex environments there will still be a need for the organisation to set standards and ensure that teams comply with them. LLMs can help to prepare materials for architectural governance meetings, but so far they are not good at designing architectures autonomously.

There may also be a need for a human to oversee the network of AI agents, ensuring they are continuously improved, as well as complying with the business's policies and standards. (Anthropic's Model Context Protocol might have a role in integrating autonomous agents.)

Agile Coaches will be important, but not in their conventional role of steering human behaviours towards agile principles. Their new role will be to redesign the development process, identifying waste and delay, and dropping agile practices if they are no longer providing value.



The traditional squad or pod will have become a Superpod. Working with code directly will have become a thing of the past. It might seem scary, but we have been here before. People used to write in assembly language – but no-one worked directly with machine code once compilers became trusted.

Bullets indicate artifacts (eg documents, deliverables, etc)

Current Adoption

The big questions are: how much of it has already happened? And how quickly will it become business as usual?

Opinions vary. A 2023 survey by Github found that 92% of developers are using AI tools at work, and more recent surveys by Insight's Amdaris software engineering division confirms this. However there are leaders and laggards.

Current Adoption

Organisations using Level 2 teams by 2025 were typically small or entrepreneurial organisations with skilled and experienced people who can fill the Product Creator role.

The obstacles are particularly visible in larger more bureaucratic organisations:

- There is a major shortage of people with the right skills to be a Product Creator;
- Some developers resist AI tooling for fear that their jobs will disappear;
- Some developers try AI but find they do not speed things up instantly. For example, this study by Thoughtworks reports that "Claude Code saved us 97% of the work – then failed utterly";
- Some organisations have policies or standards which restrict the use of generative AI;
- There are concerns about hidden costs. It's not the cost of using the LLMs. It's also that the energy costs of large scale AI will not help organisations meet their environmental goals;
- Contracts with consultancies and outsourcers are often designed to pay only for labour. They would need re-negotiation for Level 2 teams.

In 2025, there are some teams at Level 1, and a few at Level 2. Current momentum suggests that by 2030, at least half the software engineering teams in the world will be operating at Level 1.

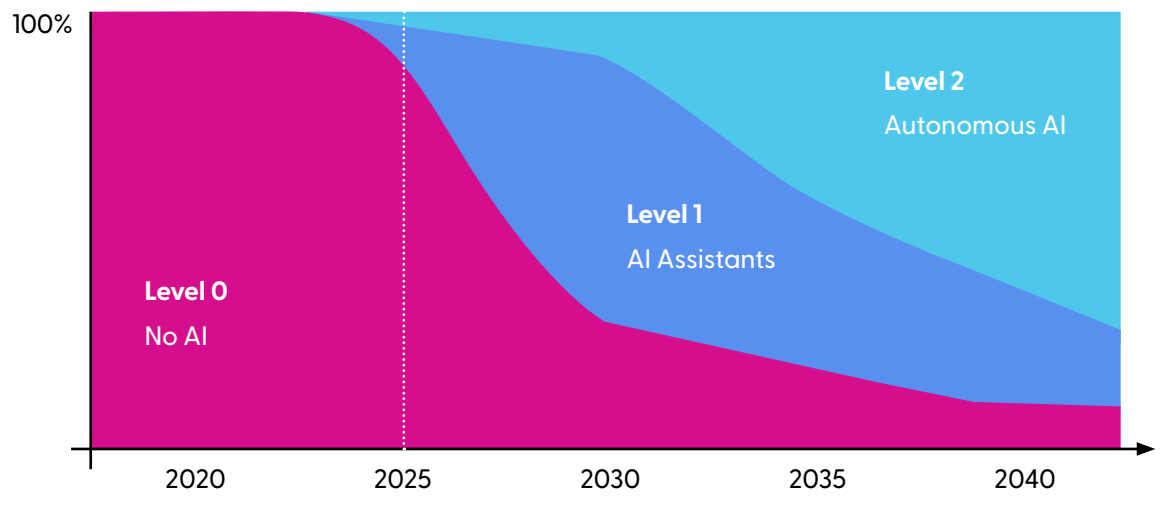
By 2040, the dominant way of working in software engineering will have become Superpods, based on Level 2 autonomous AI. Young people entering the workforce in 2040 will know no other way.

Level 0 and Level 1 software development will not disappear entirely. For example, organisations with safety critical systems will be very cautious about trusting human lives entirely to autonomous LLMs. And the history of the tech sectors shows that it takes old technologies a very long time to fully die out.

However, by 2040 we expect the software engineering industry will be dominated by Level 2 teams using autonomous AI agents.

A Forecast

Given all these factors, we expect adoption to follow something like this:



Impact

For Buyers of Software Engineering Services

For buyers of software engineering services, there are three things that should change, and one that will probably not.

1. Supplier investment in AI. Buyers should favour suppliers with an active research programme on AI-assisted software engineering, that they will share with their clients.
2. Faster delivery. Projects should be completed faster, so the time the business waits for new features should notably go down.
3. Price structure. Clients are no longer buying just labour, but a combination of labour and AI agents. The buyer's hard-earned intuition that $500/day is the market rate for a given grade and location will no longer be valid. It doesn't account for the cost of the AI agents. The good news is that the introduction of AI could stimulate better ways of contracting. The only really important measures of success are business outcomes. Suppliers who are confident in the value of AI will be more open to contracting based on value delivered rather than effort spent.

Buyers might hope that efficiency gains bring reduced spending. This is far from certain, for several reasons. AI speeds up software development in some situations, and not in others. Efficiency has long been a very difficult thing to measure in software engineering. And while less effort may be needed, there will be additional costs to pay for LLMs and ensure quality.

For Software Engineers

By software engineer here, we mean all members of a software engineering team: developers, designers, testers, scrum masters, product owners, et al.

For Senior Software Engineers, AI is an opportunity to dispense with work they don't enjoy. Senior engineers who embrace AI will find their work more enjoyable and rewarding. Those who avoid it are putting their careers at risk.

For junior software engineers, LLMs are both a gift, and a threat. They are excellent as a training aid, showing how code can be improved. It's like having a senior engineer at your side whenever you need them. However, many believe that work which in the past has been assigned to juniors can now be done by AI. Junior developers must therefore adjust their career development aspirations. The role which will be in high demand in the coming years is Product Creator, which combines technical knowledge with a deep business understanding. Technical knowledge alone will no longer be enough to sustain a career path.

On Software Engineering Service Providers

Every supplier of software engineer services is watching the development of AI intently. It is potentially a substantial threat to revenue; and at the same time an opportunity to differentiate on value delivered rather than skills and effort.

However the software engineering industry has been here before. Offshoring was seen initially as a threat, but soon became business as usual. Automated testing largely eradicated the market for manual testing services between 2015 and 2020, but created a new opportunity for QA engineering services.

Insight's Approach

Insight, as a Fortune 500 solutions integrator, is leading the charge to bring the benefits of AI to its clients.

It's software engineering division, Amdaris, is evaluating LLM tools and selecting best-of-breed. It already has many teams operating at Level 1, and is re-training engineers to develop Forward Deployed Engineers who can fill the Product Creator role. For a UK-based equipment dealer, Amdaris empowered the development team with Github Copilot to accelerate unit test creation, achieving 2–3x faster unit test writing despite the solution's complex business logic.

Insight Consulting uses AI from the first steps solving a client's business problems, and has some teams operating at Level 2. For example, at a petrochemical giant which needed to analyse images to identify safety concerns, Insight Consulting used an AI agent to read client requirements, construct a Project Initiation Document, and set up a demo within a week. 70% of the code was AI-generated. Insight Consulting is now proposing work for its clients based on value received rather than effort delivered.

MKT7603